INFORMATION STORAGE AND RETRIEVAL SYSTEMS

THESIS

Presented to the Graduate Council of the North Texas State University in Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Teresa Adams Creech, B.S.

Denton, Texas

May, 1983

This thesis describes the implementation of a general purpose personal information storage and retrieval system. Chapter one contains an introduction to information storage and retrieval. Chapter two contains a description of the features a useful personal information retrieval system should contain. This description forms the basis for the implementation of the personal information storage and retrieval system described in chapter three. The system is implemented in UCSD Pascal on an Apple II microcomputer.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. PI-STAR FUNCTIONAL DESCRIPTION</td>
<td>19</td>
</tr>
<tr>
<td>III. PI-STAR LOGIC DESCRIPTION</td>
<td>39</td>
</tr>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>A. ILLUSTRATIONS</td>
<td>45</td>
</tr>
<tr>
<td>B. PI-STAR SOURCE CODE</td>
<td>60</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>127</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

The storage and retrieval of information is not a new concept. For centuries, information has been stored in libraries and retrieved by researchers. The methods of manual cataloguing were a necessary part of the traditional library. These systems of storing and retrieving information were good, up to a limit, but as the amount of information available increased, the problems of classifying it (and later retrieving it) also increased. The estimated number of scientific journals alone existing in the years 1800, 1850, 1900, and 1966 were 100, 1000, 10000, and 100000 respectively.¹ The manual systems of classifying information could not handle the job.

As early as 1851, people were worried about the increasing amount of information available and about the problems of storing and retrieving it. John Henry, the Secretary of the Smithsonian Institute wrote that with the amount of published information available steadily increasing, there needed to be a much better method of arranging and classifying it or else...

literature and science will be overwhelmed by their own unwieldy bulk. The pile will begin to totter under its own weight, and all additions we may heap upon it will tend to add to the extension of the base, without increasing the elevation and dignity of the edifice. This warning went unheeded for years, until after World War II the flood of information became too great for existing methods of information handling to adequately take care of. The use of computers in the field of information storage and retrieval had become a necessity.

The rapid increase in the amount of information available was only one reason for this necessity. Computer systems have drastically reduced the time needed in order to gather information at a time when sources of information are world wide and information is coming in more rapidly than before. Another reason for the use of computer systems is that the changes in the quantity of information available has made it impossible for any one person to read and remember every item of interest, even in a single field. Indeed, it is economically impossible for even a single institution to process all the available information for later retrieval. With the use of computer maintained databases, any one person or institution can have access to needed information without actually owning it. This is necessary because the increasing complexity of today's society makes more people interested
in retrieving more information that they would not have been interested in before.

One of the first attempts at computerized document retrieval was done by P. R. Bagley at Massachusetts Institute of Technology. In 1951, Bagley prepared a master's thesis entitled "Electronic Digital Machines for High Speed Information Searching." In the mid-fifties, document retrieval systems were becoming more common. One of the first of a chain of document retrieval systems was done by R. H. Bracken and H. E. Tillitt at the U. S. Naval Ordnance Test Station at China Lake, California. They implemented their system on an IBM 701 computer using magnetic tape for searching.

Document retrieval systems are not the only information retrieval systems in use today. Any large business, and a good many smaller businesses, would be overwhelmed with the task of keeping track of all the information if the capability of having a computer maintained database did not exist. More and more large and small businesses are leasing computer time and/or purchasing their own large and mini computers in order to maintain the databases containing the large volume of information necessary in any business.

The type of information retrieval systems and database management systems discussed thus far have been implemented on large computers for large databases. However, with the introduction of the microcomputer, the costs of a computer
system are within reach of the average person. Numerous database management systems have been designed to meet the needs of individuals desiring to keep files of various types with the aid of a microcomputer.

One of the most complex database systems available for microcomputers is MDBS, which is marketed by Micro Data Base Systems, Inc. MDBS is a powerful database management system which combines many of the features of systems implemented on large scale computers in a format compatible with microcomputers. It implements many CODASYL supported structures such as a Data Definition Language (MDBS DDL) and a Data Manipulation Language (MDBS DMS). MDBS also supports extended network data structures and variable length records. Data security includes read/write access protection at multiple levels. A feature of MDBS that is not common to most large database management systems is the support of MANY-TO-MANY relationships. Most large systems allow only ONE-TO-MANY relationships.

Access to MDBS is possible through many host languages. Such CP/M supported languages as CBASIC2, PL/1, PASCAL Z, PASCAL M, PASCAL MT+, BDS C, CIS COBOL, and the microsoft languages COBOL, FORTRAN, or BASIC can be used to access MDBS. Other host languages include North Star BASIC, TRS Disk BASIC, and Applesoft BASIC. MDBS can be used with any type or size of disk drive.
Data records for MDBS can be extremely long, as can the fields included in the records. Practical limits, however, are up to 4K bytes for records and fields. A record can contain up to 255 fields and there can be up to 254 different types of records in a database. MDBS is designed to support multiple disk drives and the size of a database is bound only by available disk space.

Several packages are available to be used with the basic MDBS system. The first is a combination query system and report writer called MDBS QRS. This package allows the user to formulate queries to a database without writing a program. The query package implements the LIST and FIND clauses as defined by CODASYL as well as the wildcard and "match one" functions. The wildcard function matches any string of characters and the "match one" function matches any single character. The standard relational operators are also provided as are the normal boolean operators.

The report writer feature can be ignored, thus allowing the query response to be generated in a standard format or a user can design a unique report using the report writer. A report can be immediately displayed on the screen or routed to a printer. A report can also be saved on a file for later use. Some of the features of the report writer include the support of control breaks and options to generate statistics. A plan for future additions to MDBS QRS include graphical output routines.
The second package available as an extension to MDBS is a database restructuring program called MDBS DRS. This program allows a file to be restructured as needs change. With MDBS DRS a user can add, delete, or rename parts of the database. Changes can also be made in read/write access levels and passwords.

MDBS RTL, the last package available for use with MDBS, is a database recovery utility. It consists of two parts—a transaction logging facility which records all relevant database activity, and a recovery processor which uses the transaction log to update the database in the event data is lost due to an error.

The complete MDBS system, including extensions and manuals, costs less than two thousand dollars. This cost is much less than the large information retrieval systems after which MDBS is patterned. However, MDBS is one of the most expensive database systems available for microcomputers. It is also one of the most complex packages. The objective of the designers of this package was to provide "a powerful database management system that offers big-computer capability on small computers."² For the most part, they have succeeded. The only fault that a 1981 Mini-Micro Systems review could find was that MDBS did not keep a data dictionary.

and that data integrity was not tracked; rather it was left up to the user.

MDBS is a very complex package and could be very useful for the right application, but its cost is prohibitive for the average personal computer user. Also, it provides more features than the average user would need or want.

Another database system currently available is DB Master. This system is designed to be used with the Apple II Micro-computer and is much less complex than MDBS. The requirements for DB Master are an Apple II or Apple II Plus system with 48K RAM and Applesoft BASIC. The program requires at least one floppy disk drive, although two disk drives are recommended and the program supports up to four.

DB Master is one of the more complex packages available in its class (MDBS being in a class of its own). It uses a technique called "dynamic prompting." That is, the bottom two or three lines of the monitor are used to display the commands which are available to the user. The remainder of the screen is used to display queries and results. The advantage of the technique is that the commands are made known to the user at a glance; however, the forty column screen used by Apple makes many abbreviations necessary and at least one command is omitted from the list displayed.

DB Master is equipped with a three level password protection system. A password can have "read only," "read/write," or "master" authorization.
Two uninitialized diskettes are required in order to create a DB Master file. The first diskette, called the Utility Disk, holds such file information as the secondary key files and report formats. The other diskette, called the Master Disk, contains the actual file. It is possible for a file to have more than one of each type of disk, but there are typically more Master Disks than Utility Disks.

The file structure of DB Master is based on IBM's Index Sequential Access Method and allows the user of both primary and secondary keys. Additional features include a report generator and the ability to emulate a hierarchical database.

There are several different field types available for DB Master. Each field is assigned a type by the user when it is created. The field types are numeric, alphanumeric, dollar/cents, yes/no, social security number, telephone number, and date. The last three field types have a pre-defined format and any entry into these fields must have that format. Also, any field may be assigned a default value at creation.

In searching a file, DB Master allows the use of the relational operators, as well as a range operator [^], a wildcard operator [*], a "match one" operator, and the include operator (()). The include operator allows a word or phrase to be matched if it is within another phrase. The AND and OR operations are also allowed, although the manner in which the OR is implemented can be confusing.
Some of the problems encountered with DB Master were the use of the OR function and the manipulation of the screen. When the OR operator was used, the program generated a separate screen page for each OR used. Even as simple a query as "ADAMS OR CREECH" would generate two separate pages. A more complex query could easily become extremely involved.

The use of screen editing techniques in DB Master was quite good for the most part, but at points these techniques were overused and became complicated. One problem was the use of black on white lettering. Some monitors have problems with this display feature when it is used with standard sized lettering.

Each DB Master record can contain up to 1020 bytes and up to one hundred fields. There can be up to nine screen pages in a record. A single primary key can be included in a record and can consist of up to four fields. There can be any number of secondary keys.

DB Master has many nice features and would be adequate for a home-business user or a user who has need for a sophisticated database system. As with MDBS, DB Master offers too much for many personal computer users.

Create-A-Base is another database package available for the business oriented microcomputer user. It sells for under four hundred dollars, but is not as advanced as the less
expensive DB Master. It is a menu driven system which uses
prompts to aid the non-technical user. The package includes
a report generator and a sort routine as well as the main
database routines.

Create-A-Base is designed for use with Commodore com-
puters. It allows up to twenty-four variable length fields
in a record, up to 999 records in a file, and up to 220
characters per record. Some of the commands available with
Create-A-Base are Add, Create, Correct Entries, Sort, Print
Chart of Fields, Delete, Print Mailing Labels, Add Fields,
Print, Backup Disk, Generate Report, Search, and Merge.

dBASE II is a relational database package available for
around seven hundred dollars. The system operates on any
8080, 8085, or Z-80 based system running CP/M. The package
requires 48K RAM and one or more mass storage device
(normally floppy disk). A 20x80 character screen is also
required.

The dBASE II package allows queries to be formed both
interactively and through the use of programs using dBASE II
commands. The commands are simple and english-like. A
series of operations can be saved in a file to be executed at
later dates.

dBASE II allows reports to be generated in any format
the user desires. Thus, the business user can continue to
use the same forms as he did under the manual system. Totals
and subtotals can be generated for fields in a report.
Some of the commands available with dBASE II are Create, Copy (a database to a new file), Index (creates an index file of a database), Append (adds records to a database), Edit, Replace, Change, Delete (marks records for deletion), Recall (erases the deletion), Pack (deletes marked records), Select (switches between two databases in use), Sort, Find (locates an indexed record), Locate (finds records that meet specified conditions), Skip (forward or backward in the database), Do While, and If...Else.

dBASE II allows up to 65,535 records in each database file. Each record may have up to 1000 characters and up to thirty-two fields. The maximum number of characters allowed in a field is 254. Character strings, command lines, and report headers also have a limit of 254 characters.

The last database system reviewed is Request, another relational database package. It is available for the Apple II and Commodore Pet microcomputers. System requirements for the Apple II are two disk drives and 48K RAM. Requirements for the Commodore Pet are a 2040 or 8050 disk drive and 32K RAM. Also required for use with Request is "Super Kram," a keyed random access method file management program. The entire system, with Super Kram, costs around four hundred dollars. Unlike the database packages previously discussed, Request accepts input from Visicalc and The Source.
Request uses "screen masks" in the formation of queries as well as in the editing and adding of records. These "masks" are designed during the creation of the database. Queries are formed using arithmetic, boolean, and multifield comparisons. Request has the capability of building a query library, so often used queries can be executed upon demand. The report generator allows control breaks, totals, statistics, and calculations as well as other features.

The packages previously described are all database management systems. They all have varying degrees of sophistication ranging from the very complex MDBS to the simpler Create-A-Base. They are intended to be used with data which is to be manipulated. This fact is illustrated by the report generators supplied with the database management packages. The various report generators provided varying features, among them are column (or field) totals and subtotals, row totals, statistics, and even graphics. Another indication is the suggested uses of such systems. Some of the uses are: general ledger, accounts receivable and payable, journal of accounts, and other uses where the data needs to be manipulated through the use of the database management system or through auxiliary programs.

There are many applications for which the many database management systems are over qualified. Many of the applications which personal computer users wish to implement using a microcomputer system need an information storage and
retrieval system rather than a database management system. In an information retrieval system, information would be retrieved as it was sorted, without the unnecessary manipulation of the information. A good general purpose information retrieval system could be adapted to many purposes. A few possibilities include the storage of basic family tree information for the serious geneologist, an extensive recipe file for the chef, a horticulture file for the plant lover, and of course the well known bibliographic file. Many people, especially professionals, accumulate numerous periodicals. Searching such a personal library for particular articles can be tedious and time consuming. The time spent to enter articles of interest in an information retrieval file could be well worth it. Of course, the bibliographic file does not have to be limited to journal articles. A file could consist of information on books, records, tapes, or even computer programs.

As each user's needs would certainly be different, the applications of a system would vary from user to user, but a complex general purpose information storage and retrieval system could be adapted to many uses. Unfortunately such a package is not available for microcomputer users today. Several programs exist which fall into the category of information retrieval, but their numbers are far fewer than the
database systems available. Also, these programs fall far short of meeting the necessary qualifications of a good information retrieval system.

The first example of an information retrieval system was initially implemented on a Hewlett Packard System 1000 Minicomputer operating under the RTE-III Operating System. The package was written in Fortran IV. A similar package is available for the Radio Shack TRS-80 microcomputer for around one hundred dollars.

This information retrieval system has some good features. It is general, a user can design a file with records of up to eighteen uniquely named fields. The basic commands for the system are Create, Input, Query, and Stop. In the query mode, a query is made up of combinations of simple queries. A simple query is made up of a field name, a relational operator, and a value. Examples are:

\[
\text{FNAME} = \text{JONES} \\
\text{DATE} > 78/00/00
\]

Simple queries can be combined using the boolean functions AND and OR. An example of a more complex query follows:

\[
1 \text{ OR:} \\
2 \text{ AND:} \\
3 \text{ DEGREE} = \text{BS} \\
3 \text{ MAJOR} = \text{COMPUTER SCIENCE} \\
3 \text{ INTI} = \text{COMPUTER SCIENCE} \\
3 \text{ EXP1} > 3
\]

In the above example query, the two groups of level three simple queries are ANDed together by the "AND:" operator immediately above them. Then the two AND operations
are ORed together by the "OR:" command in level one. Thus, to satisfy the query, a record must contain information about a person who satisfies at least one of two requirements. He must either have a Bachelor of Science Degree with a major in Computer Science, have an interest in Computer Science and have more than one year of experience, or he must be interested in Computer Science and have more than three years of experience.

Before creating a query, the user must specify one of three options. The options are P (print the records which satisfy the query), PD (print and delete the records which satisfy the query), and D (delete the proper records without first printing them).

The only major fault found with this information retrieval system is the complete lack of editing features. Once a record is entered into the file, the contents of any of the fields cannot be changed easily. The only way to alter the contents of a record is to delete the record, then reenter it into the file along with the needed changes. The need for editing capabilities can be illustrated by the fact that no matter what care is taken, it is very easy to make typing mistakes. No provision is made for error correction as a record is entered into the system. Deleting and reentering the record only increases the chances of making more mistakes.
Another drawback, resulting from the lack of editing features, appears when an application requires that the information within a record be changed. An example is in a family tree file. A field in such a file might be "MARRIAGE DATE." The entry in this field for an unmarried family member would be "not married," but this field is subject to change. With the system being discussed, the changing of a single field involves deleting and rewriting the entire record. For certain applications, this lack of editing capability severely handicaps the information retrieval system.

The second package to be evaluated is called Information System, and it is also available for the Radio Shack TRS-80 microcomputer. This package sells for well under one hundred dollars and is described as an "in memory information system." It is designed for use in place of such manual systems as index card files or rolodex files. Suggested uses are mailing lists and inventories. It has a limited query capability and a sort routine.

The third and last information retrieval package found is designed for use with the Apple II microcomputer. The program is called Data Base Management I and is essentially an information retrieval program which includes a report writer. Commands available with Data Base Management I include Create, Select (a file), Search, Change, Enter or
Delete records, Report, Sort, and Delete the entire file. These commands are all limited in their use and the delete-file command does not work at all.

Creating a file is relatively straightforward. The user enters the file name, then the program prompts for up to twenty-one uniquely named fields to make up a record. The program creates two files--a headerfile and an indexfile. A file of all the file names is also kept by the program.

When a file is initially selected, the entire file is read into memory so the size of a file is restricted by the amount of available memory. Once a file has been read into memory, the file can be searched, but it can only be searched on one field. Also, the search is done on the beginning of the field only. Thus, the query "DATE = J" will find all records whose date fields contain anything beginning with "J." The contents of the other fields do not matter.

There is no provision made for error correction during the creation of a file or during the addition of a record. Also, there is no editing allowed on a retrieved record. The change command requires that the user enter the record number of the record to be changed. Also, only one field of a record may be edited during one change operation. The delete command also requires a record number.
Data Base Management I has very little userproofing. It is very easy to send the program into a loop from which the only means of exit is to stop the program. This fact and the limited forms of the commands makes Data Base Management I unsuitable for any serious user.

A well designed information storage and retrieval system should be a general purpose package adaptable to many varied tasks. The user of such a package should not be hampered by unreasonable restrictions such as in the number of fields per record or in the number of records in a file. The editing features of an information retrieval system are also important. These features should be complete and easy to use. The query language should be as simple as possible while allowing for complex, multifield queries. A user should be able to list (on the screen), print, edit, and delete a record retrieved as the result of a query. Above all, a good information retrieval system should be easy to use and should protect against incorrect input that could cause the program to bomb or data to be lost. The information storage and retrieval system described in the remainder of this document attempts to meet these requirements.
A useful personal information storage and retrieval system must meet requirements contained in five functional areas. First, it must be a general purpose system, adaptable to many varied uses. Second, any restrictions imposed on the system must be reasonable and necessary. Third, it is essential that the information retrieval system be easy to use. Fourth, a system must include complete editing and error correction facilities. Fifth, the query language must be able to process complex, multifield queries.

PI-STAR, a personal information storage and retrieval system, is designed to meet these five requirements.

The first requirement allows a system to meet the many varied needs of different users. Several areas combine to make an information storage and retrieval system general purpose. These features keep a system from becoming tied in to any one particular type of application.

One general purpose feature allows descriptive, user defined names for files, records, and fields. The names describe the type of data contained in the file. An example of a file with descriptive names is one containing bibliographic information. A possible file name is "LIBRARY."
A descriptive record name is "DOCUMENT;" this record could contain such fields as "TITLE," "AUTHOR," "SUBJECT," "PUBLICATION DATE," and "PUBLISHER." Another example of the use of descriptive names could be a file called "BUSINESS CONTACTS." The record name for this file might be "CONTACT" and fields could include "NAME," "STREET ADDRESS," "CITY," "STATE," "ZIPCODE," "PHONE NUMBER," "COMPANY NAME," "POSITION," "DATE LAST CONTACTED," and "NOTES."

The above examples illustrate that files can contain a variable number of fields. A system which allows this general purpose feature does not hamper a user by unnecessary fields which are required by the system. Another feature of the fields in a general purpose system is that none of the fields are explicitly typed. There are no fields designed to contain a particular type of data such as dollar amounts or dates.

There are several areas within an information storage and retrieval system which meet the second requirement, that any restrictions required by a system must be reasonable and necessary. There are many reasons why restrictions are placed upon certain aspects of a system. Hardware limitations, such as available memory space, are one reason. The programming effort involved is another. There may be constraints on program size or the coding involved in implementing a certain feature may not be worth the benefits
of including the feature. Some limitations are placed in a system in order to aid the user. A limitation which makes a system easier to use is well worth considering.

An area which may be restricted is the maximum number of fields allowed in a record. This is generally imposed by hardware restrictions on small systems, however, a limit on the number of fields isn't too restrictive if it is high enough. For example, a maximum field limit of five would be a hindrance, but an upper limit of twenty-five or more is reasonable. Limiting the number of fields is necessary in making an information storage and retrieval system easier to use. For example, a system may be designed to accept single character field identifiers instead of the longer field names. This feature requires limiting the number of fields because a set of unique field identifiers must be defined. The benefits of this feature are that it lessens the likelihood of typing errors and it requires fewer keystrokes. The only disadvantage is that the identifiers do not describe the field, but a user soon learns what the identifiers are in an often used file. Also, commands are provided which list or print the field names and their associated identifiers.

The maximum length of a field may also be limited, but there must be enough room for the data to be entered without abbreviations and limiting of data. A maximum field length
of twenty characters is much too restrictive. A more reasonable upper limit is .5K characters.

The file creation process of a good information storage and retrieval system must not restrict the maximum length of the names of files, records, and fields. These names must be long enough to contain complete and descriptive names. For example, the name "LIBRARY" is more descriptive than the name "LIB." In order to be able to create descriptive names, all alphanumeric characters, plus the underscore must be allowed.

One area in an information storage and retrieval system which must not be limited is the file size. The maximum number of records in a useful file is virtually unlimited. The reason for this is that, for most applications, the maximum file size cannot be determined. A file must be able to grow as an application does and limiting the number of records can make an otherwise useful information storage and retrieval system impossible to use.

The third functional area requires that a good information retrieval system be designed to be as simple as possible to operate. A system which is complicated and difficult to use can be very frustrating. This lessens the usefulness of the system. There are many things which can be done to make a system easy to use.
Clear and understandable instructions and prompts are features which make a system easy to use. Messages in appropriate places also help. An example of a useful message is the displaying of all system commands available at any given time. This gives the user immediate knowledge as to what is and isn't allowed. The use of simple, short keying sequences is another feature which make a system easy to use. For example, a "Q" is easier to type than the word "QUERY."

A large amount of userproofing is necessary in a good information retrieval system. Userproofing is protection against typical user errors. Built in safeguards protect against possible serious consequences of these errors. One safeguard which protects the user is an exit option at the beginning of critical system actions, thus preventing the execution of a routine by mistake. Procedures must also contain mechanisms to interrupt any command during execution, except in instances where interruption would destroy important information.

Proper editing and error correction features are necessary in any good system. Editing is the changing of data within an already existing record. Error correction allows the user to correct errors before adding new information to a file.
Any record which is retrieved from a file must be able to be edited. The ability to edit a record allows correction of input errors not discovered when a record was created. It also allows updating of the information contained in a record. An information retrieval system lacking in editing features is all but useless except for the most static files and the most perfect typist.

The error correction process is an essential step any time new information is added to a file. It helps insure that a new record is error free before it is added to the file. Similar error correction is also necessary after a record has been edited and when a file is being created. The error correction routines of a useful system can be executed many times as necessary to correct errors.

Record deletion is related to editing and error correction. It is sometimes necessary to remove outdated or unwanted records. These records, if not deleted, can cause confusion if they are retrieved. A complete information storage and retrieval system allows any record to be removed if necessary.

The fifth and most important feature of a good information storage and retrieval system is the query language. A query language must be capable of handling complex, multifield queries. A query language with limited search capability, such as allowing searches based on only one
field, is almost useless. Because of the complex queries which can be handled, the query language must be as clear and concise as possible.

PI-STAR is a personal information storage and retrieval system which meets the above requirements. The remainder of this chapter gives a detailed description of PI-STAR.

PI-STAR is a software system designed by the author. It is composed of a set of procedures which create, search, and modify a file specified by the user.

Certain PI-STAR routines are automatically entered when PI-STAR is first activated. These routines form the base of the entire information storage and retrieval system. The other PI-STAR procedures are called from this initial state and control returns here when a procedure is completed.

The first action taken by these routines at start-up time is to prompt for the name of the file to be accessed. This information is necessary for any file, whether it is an existing file or one which is to be created. If the file already exists, only the first ten alphabetic characters of the file name need to be typed. A total of thirty characters are allowed in the file name. If a null file name is entered, program execution is immediately terminated. After the file name is entered, a menu is displayed which shows
all the commands which are available from the initial state.

The PI-STAR start-up menu looks like this:

SELECT:
    C)REATE FILE
    A)DD RECORDS
    S)EARCH FILE
    D)ELETE MARKED RECORDS
    E)XIT PI-STAR

To select an option, the user simply enters the first letter of one of the menu items. This menu format is used throughout the PI-STAR system and aids in making PI-STAR easier to use.

There are several other features which are used throughout PI-STAR to make the system easier to use. For example, at the beginning of the execution of any option, PI-STAR asks if the user wishes to continue with the execution of the option or return to the start-up menu. This protects against possible disasters--such as executing the delete option on a file in which the marked records are not yet ready for deletion. Another feature used throughout PI-STAR prevents information from being removed from the screen before the user has finished reading it. When PI-STAR fills up the monitor screen, it issues a message and waits for the user to signal the system to continue. After about five seconds of inactivity, the system continues automatically. If the user enters an escape ('esc') character, the system continues immediately. If any other key is entered, a permanent wait state is entered which is terminated by the user typing any key.
The PI-STAR editor is used to input and correct new records, edit existing records, and input and correct queries. The editor provides the ability to move the cursor anywhere in the text (up, down, right, left, and carriage return), to add characters, and to erase characters. A total of 492 characters may be input on one screen. This figure does not include any carriage returns generated. The data is entered on lines which can be up to eighty characters in length. Up to twenty lines of varying length can be input. If the eightieth character is entered on a line during an add, the editor generates a carriage return. A carriage return cannot be generated if there are characters on the line to the right of the cursor.

To erase characters, the cursor must be positioned immediately to the right of the first character to be deleted. After the 'E' is typed to enter the erase mode, the left arrow key erases characters and the right arrow key restores them. Characters can be erased back to the beginning of the line and restored to the point where 'E' was entered.

The bottom of the edit screen contains four lines of information to aid in editing. The first line marks every ten columns with a number from one to eight. The second line displays a message indicating the current edit mode: Add, Erase, or Move Cursor. The commands available in the
current mode are displayed in the third line and the fourth line contains a message. The messages differ depending on what type of edit is being done. For example, when a query is being corrected, the message gives an indication of what type of error was found.

The process to create a file is initiated when the "create file" option is selected from the start-up menu. The purpose of the create process is to gather essential file information and set up the basic file structure. The information gathered includes the name of the file, the record name, the number of fields, and the field names. The file name used is the one which was entered at the beginning of the PI-STAR session, so the first information prompted for by the create process is the record name. The plural form of the record name is also asked for. It is used to make messages more readable. When the record name has been entered, PI-STAR displays the file name and both forms of the record name. The user can then make corrections to these names. After the record name has been entered and corrected, the system prompts for the field names.

As each field is entered, it is assigned a unique, one character field identifier. These identifiers are used by other parts of the PI-STAR system instead of the actual field names. The field names are actually comments, and so
can contain any characters, even blanks. Each field name can contain up to thirty characters. The thirty-six field identifiers allowed by PI-STAR consist of the letters A-Z (note that the identifiers must always be entered in upper case) and the numbers 0-9. The system prompts for the field names by displaying these field identifiers. The field name is entered after each identifier. Field input is terminated when an empty field is entered or when 36 fields have been input. The create process is illustrated in figure 1 (see Appendix A).

Error correction is allowed on the field as it is being entered. Further error correction is allowed after the field has been entered. After all existing fields have been input and corrected, the user is given the opportunity to add more fields which may have been omitted. When the user indicates that the information is correct, the file is created.

A newly created file is empty until the add procedures are executed. These procedures are invoked by selecting the "add record" option from the start-up menu. The add routines can be called anytime one or more new records are to be written into a file.

After a field has been entered, the PI-STAR system lists the field. It then allows the user to correct errors and, when the user indicates that it is error free, the
next field is input. The input and correct cycle continues until an entire record has been input correctly. The record is then added to the file. More than one record can be added during an add session. PI-STAR is specifically designed so there is no limit on the total number of records which can be contained in a PI-STAR file. An example of part of the add process is shown in figure 2 (see Appendix A). In this example, the last field of a record is being displayed for correction, then the record is written to the file and PI-STAR asks if there is another record to be input.

In order to enter the routines which allow the user to form a query and search a file, the "search file" option of the start-up menu must be selected. As the search section of PI-STAR is entered, the following menu is displayed:

```
SELECT:
   I)NPUT QUERY
   S)EARCH MENU
   P)RINT QUERY
   D)ISPLAY FIELD NAMES AND IDS
   O)UTPUT FIELD NAMES AND IDS
   E)XIT
```

The commands "input query," "print query," "display field names and identifiers," and "print field names and identifiers" can be executed as many times as necessary during query input. The print and output commands work only if there is a printer on-line. The display command outputs
a list of the field names and their identifiers on the monitor. The "input query" command enters the editor, where a query may be input or edited. The "exit" command causes a return to the main menu. The "search menu" command causes the query to be translated and the search menu to be displayed.

The building block of a query is the simple relational expression. A simple expression is made up of a field identifier, a relational operator, and a search string. An example is "A = 'ADAMS'." The field identifier (A) indicates which field of a record is to be searched. The search string (ADAMS) identifies what the PI-STAR system is to attempt to locate. The search string must be enclosed in either single or double quotes. The six relational operators (= in the example) define the type of search to be performed on the field.

Valid relational operators are "equal to" (=), "not equal to" (<>), "less than" (<), "less than or equal to" (<=), "greater than" (>), and "greater than or equal to" (>=). The "equal to" and "not equal to" relational operators search anywhere in the indicated field. If, when using the "equal to" relational operator, a word or series of words are found which exactly match the search string, the search is successful. The "not equal to" search is successful when the word or words contained in the search strings
are not found in the field being searched. The other relational operators cause searches from the beginning of the indicated field. The search string is compared to an equal number of characters contained at the beginning of the field. A match occurs when the beginning of the field is "less than," "greater than," "less than or equal to," or "greater than or equal to" the search string, depending upon the operator used.

The wildcard operator (*) can be used in any search string which is used with the "equal to" and "not equal to" relational operators. The wildcard operator matches zero or more characters and allows a search to be made for a word or words which contain certain specified characters, but which may contain other unknown characters. For example, the simple expression "A = DAN*" matches such strings as "DAN," "DANNY," "DANIEL," "DANTE," and "DANGER." The wildcard operator can be used more than once in a search string and is useful for looking for a general group of related items or for information the user cannot remember precisely. PI-STAR allows the use of the three logical operators: "and," "or," and "not." The legal form of these operators is: /AND/, /OR/, and /NOT/. Simple relational expressions are linked together with the logical operators "and" and "or" to form complex queries. The "and" and "or" operators allow searches on several fields to be specified in a single query.
Expressions which are anded together must both be successful in order for the entire expression to be considered successful. The "or" operator indicates that at least one of the simple expressions must be true in order for the entire expression to be considered successful. The "not" logical operator allows a portion of a query to be successful if its requirements are not met.

Because of their complexity, queries can rapidly become confusing. In order to make a query clear and easy to follow, spaces and carriage returns are allowed at many points within a query. Another feature allowed by PI-STAR is the use of parentheses. Parentheses can be used, as in mathematics, to clarify a complex expression or to impose a non-standard precedence.

When a PI-STAR query is evaluated, the six relational operators are evaluated first. The next operator to be evaluated is the "not" operator followed by the "and" and "or" operators, in that order. This operator precedence can be modified by the use of parentheses, because expressions within parentheses are evaluated before the remainder of a query. The query in figure 3 (see Appendix A) retrieves records which meet two requirements. First, the author (A) field must contain a name which begins with "TO" and ends in "N." Second, the title (B) field must contain one of the following titles: "THE HOBBIT," "THERE AND BACK AGAIN," or "THE LORD OF THE RINGS."
Once the query is entered and the "search menu" option is selected, the PI-STAR system checks for correct syntax. If a syntax error is detected by the system, the editor is entered and an appropriate error message is displayed. The query is redisplayed for correction. If the syntax is correct, the query is translated into a form usable by PI-STAR. Once the translation is a success, the search menu is displayed. The menu looks like this:

SELECT:

B)EGIN SEARCH
L)IST RECORD
P)RINT RECORD
E)DIT RECORD
M)ARK RECORD FOR DELETION
R)ECALL MARKED RECORD
A)BORT CURRENT SEARCH
T)ERMINATE SEARCH MODE

During a search, the query is compared with each record of a file, one by one. If the record meets the query's requirements, it is retrieved for user manipulation. When a record does not meet the requirements of a query, the system continues to the next record.

When a record is retrieved, the search menu is again displayed unless the response to the question "PRINT ALL RETRIEVED RECORDS?" was yes. In this case the record is automatically printed and the search continues without returning to the menu. The functions which are used to manipulate a record include listing, printing, and editing the retrieved record. The select menu also contains functions which aid the search process without directly
affecting the record. A menu option is selected by entering the first letter of the command name. Figure 4 shows a retrieved record (see Appendix A). This record was retrieved because it matched the query in figure 3 (see Appendix A).

The first of the commands which manipulate the current record, the "list" option, allows the user to view the retrieved record by displaying it on the monitor. The file and record names are displayed, followed by each field name and the data stored in each field. The "print" command is identical to the list command except that a permanent copy of the record is generated on a printer. The print and list commands can be executed any number of times for each record retrieved.

The "mark" command labels the record as requiring deletion. This command does not actually cause the deletion of a record, it simply places a unique identifier in a record which allows a garbage collection routine to determine which records to delete. This allows the actual deletion to be done at the users discretion, after he has had a chance to determine whether a record was really intended for deletion. Marked records are no different from any other records in that they are retrieved if they meet the requirements of a query. When a record is retrieved, the first field displayed indicates whether or not a record is
marked for deletion. An 'N' means that the record is not marked; an 'M' means that a record is marked.

The field identifier "#" is used to deliberately search for records which are marked or to prevent marked records from being retrieved. One use of this feature is to check each marked record to be sure that the record should actually be deleted. The following example shows a search for marked records:

\[(A = 'Douglas' \text{ OR} A = 'David') \text{ AND} # = 'M'\]

This query will prevent any unmarked records which satisfy the rest of the query from being retrieved.

The "recall" command is the companion of the mark command. It removes the indicator which marks a record for deletion. This action corrects the possible error of a record being marked when it shouldn't be deleted.

The "edit" command is the last command which manipulates the current record. It allows the record to be modified and then stored back into the file in its new form. When the edit routines are executed, the user is asked to enter the identifiers of the fields which need changing. Each of these fields is displayed and the user is allowed to make the changes. The PI-STAR edit command allows any number of the fields in the current record to be edited during any one edit session.
The remainder of the commands included in the select menu do not interact directly with the retrieved record. Instead they work upon the search environment. This set of commands determines whether or not a search is to be continued and allows access to query information.

The "begin search" command starts a search or resumes the current search. It is used after a record has been retrieved and, perhaps, manipulated. The "abort" command causes the current search to be abandoned and returns to the prompt to enter a query. Like the abort command, the "terminate" command also halts the current search. The terminate command, however, causes an immediate exit from the query processing routines.

The PI-STAR system includes one utility. The garbage collection utility deletes those records which had previously been marked. The utility is included in the start-up menu and can be executed whenever the user wishes.

The garbage collection utility removes records which have been marked and frees up the space occupied by the record for use by the PI-STAR system. Once the garbage collection routines have been run on a file, the marked records are actually deleted and cannot be accessed in a search.

The personal information storage and retrieval system described is aimed for use with any of the personal computer
systems available, yet it is a complex system which meets all the requirements of a personal information storage and retrieval system. It is general purpose, contains no unreasonable restrictions, is easy to use, allows complete editing and error correction facilities, and implements a complex query language.

PI-STAR is implemented in UCSD Pascal on an Apple II microcomputer. The system requires two floppy disk drives and the Apple II Language Card System (in order to have the use of the Pascal Operating System). A printer is optional, but recommended. The implementation of the PI-STAR system follows.
CHAPTER III

PI-STAR LOGIC DESCRIPTION

The PI-STAR system consists of three programs and nine Pascal units. Because of PI-STAR's size, only one of the programs and the units it needs are in memory at any one time. The programs use the Apple Pascal chaining facility to transfer from one program to another.

The three programs are MAIN, MAINSRCH, and MAINUTL. MAIN is the entry point of the PI-STAR system. MAINSRCH calls the procedures to input a query and to search a file. MAINUTL contains the delete procedures. Figures 6, 9, and 14 (Appendix A) show the structure of each of the programs.

When MAIN is executed, it outputs the title page, prompts for the name of the file to be accessed, and displays the main menu. MAIN then prompts for input from the user. If a 'C' is input, MAIN calls the CREATEFILE procedure. If an 'A' is input, the ADD procedure is called. A 'D' or an 'S', when input, causes MAIN to chain to either MAINUTL or MAINSRCH. PI-STAR execution is terminated when an 'E' is entered. The overall structure of PI-STAR is shown in figure 5 (see Appendix A).
MAINSrch is chained to when the search option of the main menu is selected. It calls the procedure QUERYMAIN to input a query and search a file. After control returns to MAINSRCH from QUERYMAIN, MAINSRCH chains back to MAIN.

Chaining to MAINUTL occurs when the delete option of the main menu is specified. MAINUTL calls KRUNCH, a procedure which deletes all the records which are marked with a "P" or an "M". A "P" means that the existing record was edited and the new version was rewritten to the end of the file. An "M" means the record was marked for deletion. After the delete is finished, MAINUTL chains back to MAIN.

The nine units which make up the remainder of PI-STAR are GLOBAL, SUPPORT, EDIT, CREATE, ADDREC, SQUERY, QRYINPUT, TRANSLATE, and EVALUATE. GLOBAL contains declarations and initializations which are global to the PI-STAR system. Every other unit and all three programs use global. SUPPORT contains service procedures which are used throughout the PI-STAR system, so SUPPORT is used by most of the other units and programs in PI-STAR. The structure of each unit, except GLOBAL is shown in figures 7, 8, 10, 11, 12, 13, 15, and 16 (see Appendix A).

The EDIT unit contains the code which makes up the PI-STAR editor. Other parts of PI-STAR call the procedure EDITOR when there is a field or a query to be input or corrected. The procedures UP, DOWN, RIGHT, LEFT, and
RETURN provide the ability to move anywhere within the text being edited. ADDCHAR and INSERT allow characters to be added to the text, and ERASECHAR deletes characters from the text. LEFTARROW is called from ADDCHAR and ERASECHAR. It causes the cursor to backspace, removing characters as it goes. RIGHTARROW causes the cursor to space forward, restoring characters removed by LEFTARROW in ERASECHAR.

The CREATE unit contains the code to set up a new file. The procedures in CREATE check to make sure that a file with the same file name doesn't already exist. If the file doesn't exist, CREATE prompts for the singular and plural forms of the record name, and allows the file and record names to be corrected. CREATE then continues to the field names. After the field names are input, and corrected, CREATE calls the DISKADD procedure from SUPPORT to create the PI-FILE header and data files.

The ADDREC unit allows the user to add records to a file or to initialize a new disk to an existing file. When adding records ADDREC calls the EDITOR. The EDITOR displays a message indicating what field is being input. When the EDITOR is exited, ADDREC displays the field just entered and if the user indicates that the field is not correct, the EDITOR is called again. This process is repeated until the field is correct. At that point, ADDREC continues to the next field. The ADDREC procedures create a temporary file
on the PI-STAR disk for each of the fields entered. After all the fields have been input, ADDREC calls the procedure WRITEFILE (from the SUPPORT unit) to add the record to the file.

When adding a new disk, ADDREC first checks to be sure that an existing file won't be destroyed when the new disk is initialized, then the DISKADD procedure from SUPPORT is called to set up the new disk. A header file is created which contains such file information as file name, record names, and file names. An empty data file is also created. The WRITEFILE procedure writes records to this file.

The SQUERY unit allows files to be searched and retrieved records to be manipulated. The main procedure of SQUERY, QUERMAIN, first calls INPUTQUERY and, when a legal query has been entered, calls QUERYSEARCH. Queries can be input and searches made until the user exits the search procedures with the Terminate command.

The QRYINPUT unit contains the INPUTQUERY procedure called from SQUERY. The QRYINPUT procedures call the EDITOR to allow a query to be input. After a query has been input, the procedure TRANS, from the TRANSLATE unit, is called to translate the query into a form which can be used by the search procedures.

QRYINPUT contains three additional procedures which aid the user. PRTNAM prints the name and identifier of each
field or issues an error message if there is no printer on-line. DSPLYNAM displays the same information on the monitor. PRTQUERY prints the query or issues an error message if there is no printer on-line.

The TRANSLATE unit's main procedure, TRANS, calls TOKENIZE, SYSNTAXCHECK, and POSTFIX. TOKENIZE translates the original query into two tokenized forms of the query. The generic form of the query (i.e., one which only identifies the token by its type, not by the particular item it happens to be) is passed to SYSNTAXCHECK. If the query gets through the syntax checking process without an error, the complete tokenized version of the query is passed to POSTFIX where it is translated into postfix (Polish) notation. If an error is detected during any stage in the translation process, then the EDITOR is called and an appropriate error message is displayed. The user can correct the query and enter the translate process again or he can exit.

The EVALUATE unit takes the postfix form of the query and compares it to a record in the file. EVALUATE returns either true or false, depending on whether or not the record satisfied the query. The EXPREVAL procedure removes a token from the postfix form of the query. If the token is an operand, it is pushed onto a stack. If the token is an operator, it must be evaluated. EXPREVAL pulls the proper number of operands off the stack and evaluated the resulting expression. If the operator is a relational operator,
RELEXPREVAL is called to evaluate the relational expression. The result of any expression evaluation is pushed onto the stack. When the evaluation is complete, the stack contains one value, either true or false, which is the final result.

RELEXPREVAL retrieves a field from the record being searched and calls either EQNE (for Equal to or Not Equal to relational operators) or LELTGEQT (for the other relational operators). These procedures evaluate the expression and return either a true or false.
APPENDIX A

ILLUSTRATIONS
WHAT IS THE NAME OF A SINGLE RECORD?
BOOK
WHAT IS ITS PLURAL?
BOOKS

FILENAME: LIBRARY
RECORD NAME(SINGULAR): BOOK
RECORD NAME(PLURAL): BOOKS

ARE THESE NAMES CORRECT?
->Y

ENTER FIELD NAMES
<RETURN> TO TERMINATE INPUT:
A) TITLE
B) AUTHOR
C) SUBJECT
D)

PLEASE CHECK THE FOLLOWING
FIELD NAMES FOR ERRORS:
A) TITLE
B) AUTHOR
C) SUBJECT

ARE THE FIELD NAMES CORRECT?
->Y

ARE THERE ANY FIELDS TO BE ADDED AFTER C?
->N

PLEASE WAIT -
FILE BEING CREATED

Figure 1. Create
FIELD AUTHOR:
ISAAC ASIMOV

IS THE FIELD CORRECT?
->Y

RECORD READY TO BE WRITTEN TO FILE
TYPE <SPACE> TO CONTINUE
<E> TO ABORT ADD

PLEASE WAIT -
WRITING RECORD TO FILE

ADD ANOTHER RECORD?
->Y

---------------------------------------------------------------
underscored characters are user input

Figure 2. Add

A = "TO*N" /AND/
(B = "THE HOBBIT" /OR/
B = "THERE AND BACK AGAIN" /OR/
B = "THE LORD OF THE RINGS")

---------------------------------------------------------------
Figure 3. Query

RECORD MARKED: N
AUTHOR:
J. R. R. TOLKIEN
TITLE:
THE HOBBIT
SUBJECT:
FANTASY

---------------------------------------------------------------
Figure 4. Retrieved Record
Figure 7. Create Unit
Figure 8. Addrec Unit
Figure 9. Mainsrch Program
Figure 10. Squery Unit
Figure 11: Query Unit
Figure 12. Translate Unit
Figure 14. Mainutl Program
Figure 15. Support Unit
Figure 16: Edit Unit
APPENDIX B

PI-STAR SOURCE CODE
**PI-STAR MAIN PROGRAM**

**PROCEDURE SETUP:**

(*PI-STAR initialization and prompt for file name*)

```
VAR
ISTRING : STRING[30]; (* TEMP STRING *)
BEGIN
(* SYSTEM INITIALIZATION STUFF HERE *)
PAGE(OUTPUT); LINECOUNT := 0;
GETVAL(ISTRING);
FNAME := ISTRING;
IF LENGTH(FNAME) = 0 THEN BEGIN
  GOTOXY(0, 2);
  WRITELN(OUTPUT,'PI-STAR:26');
  GOTOXY(C),6);
  WRITELN(OUTPUT,'PERSONAL':24);
  WRITELN(OUTPUT,'INFORMATION STORAGE AND RETRIEVAL SYSTEM':40);
  GOTOXY(0,11);
  WRITELN(OUTPUT,'COPYRIGHT 1982':27);
  WRITELN(OUTPUT,'BY':21);
  WRITELN(OUTPUT,'TERESA A. CREECH':28);
  GOTOXY(0,19);
  WRITELN(OUTPUT,'PLEASE ENTER THE NAME OF THE FILE':33);
  WRITELN(OUTPUT,'TO BE CREATED OR ACCESSED':33);
  WRITE(OUTPUT,'->');
  FILENAME := NULL;
  GETFNAME;
  END
ELSE GETHEADER;
END; (* SETUP *)
```

**PROCEDURE SELECT:**

(*Contains select menu, calls routines according to user input*)

```
VAR
  ANSWER : CHAR; (* USER SINGLE CHAR INPUT *)
  IORES : INTEGER; (* IORESULT *)
  OKSET : SETOFCHAR; (* LEGAL USER INPUT *)
  TSTRING : STRING[30]; (* TEMP STRING *)
BEGIN
  REPEAT
    PAGE(OUTPUT); LINECOUNT := 0;
```
GOTOXY(0,2);
WRITELN(OUTPUT,"P I - S T A R":126);
GOTOXY(0,7);
WRITELN(OUTPUT,"ACTIVE FILE: ",FILENAME);
GOTOXY(0,10);
WRITELN(OUTPUT, 'SELECT':)
WRITELN(OUTPUT, 'C)REATE FILE':19);
WRITELN(OUTPUT,'A)DD RECORDS':19);
WRITELN(OUTPUT,'S)EARCH FILE':19);
WRITELN(OUTPUT,'D)ELETE MARKED RECORDS':29);
WRITELN(OUTPUT,'E)XIT PI-STAR':20);
WRITELN(OUTPUT);
WRITE(OUTPUT, '->');
OKSET := ['A','a','C','c','D','d','E','e','B','b','S','s'];
GETANSWER(OKSETANSWER);
WRITE(OUTPUTANSWER);
IF NOT(ANSWER IN ['E','e']) THEN CASE ANSWER OF
'C': CREATEFILE;
'A', 'A': BEGIN
(* CHECK FOR FILE EXISTING *)
CHKFILE(IORES);
IF IORES = 0 (* FILE FOUND *) THEN ADD;
END;
'S', 'S': BEGIN
(* CHECK FOR FILE EXISTING *)
CHKFILE(IORES);
IF IORES = 0 (* FILE FOUND *) THEN BEGIN (* FILE FOUND *)
TSTRING := FNAME;
SETCVAL(TSTRING);
SETCHAIN('PI-STAR:MAINSRCH');
ANSWER := 'L';
END;
END;
'D', 'D': BEGIN
(* CHECK FOR FILE EXISTING *)
CHKFILE(IORES);
IF IORES = 0 (* FILE FOUND *) THEN BEGIN (* FILE FOUND *)
TSTRING := FNAME;
SETCVAL(TSTRING);
SETCHAIN('PI-STAR:MAINUTIL');
ANSWER := 'L';
END;
END;
END; (*CASE*)
UNTIL ANSWER IN ['E','e','L'];
(* RESET CVAL IF EXITING PI-STAR *)
IF ANSWER IN ['E','e'] THEN BEGIN
TSTRING := NULL;
SETCV(tSTRING);
END;
END; (* SELECT *)

BEGIN (* PI-STAR MAIN PROGRAM *)
SETUP;
    IF FILENAME <> NULL
    THEN SELECT;
    PAGE(OUTPUT);
END.

(*$S+*)
(*(*$L PRINTER:* *)

PROGRAM MAINSRCH;
USES APPLESTUFF, CHAINSTUFF, GLOBALSUPPORT,
    EDIT, TRANSLATE, EVALUATE, ORYINPUT, SQLQUERY;

VAR
    TSTRING: STRING[80]; (* TEMP STRING *)
BEGIN
    GETCV(tSTRING);
    FNAME := TSTRING;
    QUERYMAIN;
    SETCHAIN(PI-STAR:PI-STAR);
END.
PROGRAM MAINUTL;
USES CHAINSTUFF,APPLESTUFF,GLOBAL,SUPPORT;

VAR
ANSWER : CHAR; (* USER INPUT *)
DELCNT : INTEGER; (* NUMBER OF RECORDS DELETED *)
OKSET : SETOFCHAR; (* LEGAL INPUT *)
TSTRING : STRING[80]; (* TEMP STRING *)

PROCEDURE CRUNCH;
(* CRUNCH REMOVES ALL RECORDS WITH # FIELD *)
(* VALUES OF 'P' AND 'M', *)
VAR
ANSWER : CHAR; (* USER INPUT *)
COUNT : INTEGER; (* LOOP COUNT *)
IRESULT : INTEGER; (* IORESULT *)
OKSET : SETOFCHAR; (* OK INPUT *)
TEMPCHAR : CHAR; (* TEMP CHAR *)
WAITCOUNT : INTEGER; (* WAIT *)
WHOLEFILENAME : STRING[23]; (* FILE NAME *)

PROCEDURE FINDREC;
BEGIN
(* CHECK DFILE UNTIL A RECORD IS FOUND TO KRUNCH *)
(* OR UNTIL THE END OF THE FILE IS REACHED. *)
READLN(DFILE,TEMPCHAR);
WHILE (TEMPCHAR = 'N') AND NOT(EOF(DFILE)) DO BEGIN
READLN(CFILE);
FOR COUNT := 1 TO FIELDCNT DO BEGIN
READLN(DFILE);
READLN(CFILE);
END;
READLN(DFILE,TEMPCHAR);
END;
END; (* FINDREC *)

PROCEDURE DELREC;
BEGIN
(* KRUNCH 'P' AND 'M' RECORDS UNTIL EOF *)
WHILE NOT(EOF(DFILE)) DO
BEGIN
  IF TEMPCHAR = 'N'
  THEN BEGIN (* COPY RECORD TO CFILE *)
    WRITELN(CFILE, TEMPCHAR);
    FOR COUNT := 1 TO FIELDCNT DO
      BEGIN
        WHILE NOT(ENDL(DFILE)) DO
          BEGIN
            READ(DFILE, TEMPCHAR);
            WRITE(CFILE, TEMPCHAR);
          END;
        READLN(DFILE);
        WRITELN(CFILE);
      END;
  END;
  ELSE BEGIN (* SKIP OVER RECORD *)
    FOR COUNT := 1 TO FIELDCNT DO
      READLN(DFILE);
    DELCNT := DELCNT + 1;
  END;
END; (* DELREC *)

BEGIN
  PAGE(OUTPUT);
  LINECOUNT := 0;
  GOTOXY(0,3);
  WRITELN(OUTPUT, '"DELETE':27);
  GOTOXY(0,20);
  WRITELN(OUTPUT, 'TYPE SPACE TO CONTINUE':29);
  WRITELN(OUTPUT, '<E> TO EXIT DELETE':28);
  OKSET := {'E', 'e', SPACE};
  GETANSWER(OKSET, ANSWER);
  IF ANSWER = SPACE
  THEN BEGIN
    GETHEADER;
    WHOLEFILENAME := CONCAT('PI-FILE:', FNAME, '.DATA');
    RESET(DFILE, WHOLEFILENAME);
    RESET(CFILE, WHOLEFILENAME);
    PAGE(OUTPUT);
    LINECOUNT := 0;
    GOTOXY(5,12);
    WRITELN(OUTPUT, '*********WARNING**********');
    WRITELN(OUTPUT);
    WRITELN(OUTPUT, 'DELETE IN PROGRESS. DON'T OPEN DRIVES');
    DELCNT := 0;
    FINDREC;
    DELREC;
    CLOSE(DFILE, NORMAL);
    CLOSE(CFILE, CRUNCH);
    WRITELN(OUTPUT);
    WRITELN(OUTPUT);
    WRITELN(OUTPUT, 'DELETE FINISHED');
WRITELN(OUTPUT);
IF DELCNT = 1
  THEN WRITELN(OUTPUT, DELCNT, ', RECHAMSING, ' DELETED');
ELSE WRITELN(OUTPUT, DELCNT, ', RECHAMPLUR, ' DELETED');
WRITELN(OUTPUT);
WRITELN(OUTPUT);
WRITE(OUTPUT, 'TYPE <SPACE> TO CONTINUE');
OKSET := (SPACE);
GETANSWER(OKSET, ANSWER);
END;
END; (* KRUNCH *)

BEGIN (* PI-STAR MAIN UTILITY PROGRAM *)
GETCVAL(TSTRING);
FNAME := TSTRING;
KRUNCH;
SETCHAIN('PI-STAR:PI-STAR');
END.
UNIT CREATE;
  (* ACCEPTS DATA, ALLOWS CORRECTION OF DATA, *)
  (* THEN CREATES FILE BASED ON DATA *)

INTERFACE

USES APPLESTUFF,GLOBAL,SUPPORT;

PROCEDURE CREATEFILE;

IMPLEMENTATION

PROCEDURE ADDFLDNAMS;
  (* INPUTS FIELD NAMES *)
VAR
  LOCSTR : STRING[30]; (* TEMPORARY STRING *)
  NEXTFIELD : INTEGER; (* NEXTFIELD TO BE ENTERED *)
BEGIN
  WRITELN(OUTPUT); PAGING;
  WRITELN(OUTPUT,'ENTER FIELD NAMES'); PAGING;
  WRITELN(OUTPUT,'<RETURN> TO TERMINATE INPUT:'); PAGING;
  NEXTFIELD := FIELDCNT + 1;
  REPEAT (*UNTIL NEXTFIELD >36 OR FIELDNAME IS NULL *)
    WRITE(OUTPUT,FIELDID[NEXTFIELD],:')';
    READLN(INPUTLOCSTR); PAGING;
    IF LOCSTR <> NULL THEN BEGIN
      FIELDNAM[NEXTFIELD] := LOCSTR;
      NEXTFIELD := NEXTFIELD + 1;
    END;
  UNTIL (NEXTFIELD >36) OR (LOCSTR = NULL);
  FIELDCNT := NEXTFIELD - 1;
END; (* ADDFLDNAMS *)

PROCEDURE FLDNAMFIX;
  (* ALLOWS USER TO CORRECT FIELD NAMES *)
VAR
  ANSWER : CHAR; (*USER SINGLE CHAR INPUT *)
  CORRECT : INTEGER; (*IF YES - DATA OK *)
  COUNT : INTEGER; (*LOOP COUNTER *)
  ERROR : INTEGER; (*FLAG. ERROR IN INPUT *)
  ERRORID : CHAR; (*FIELD TO BE CORRECTED *)
  ERRORIDS : STRING[62]; (*FIELDS TO BE CORRECTED *)
  FIELDNO : INTEGER; (*CURRENT FIELD *)
  INFO : STRING[50]; (*TEMP FIELDNAME INPUT *)
  NEWFLDS : INTEGER; (*NUMBER OF NEW FIELDS *)
  OKSET : SETOFCHAR; (*LEGAL INPUT CHAR S *)
BEGIN
CORRECT := NO;
REPEAT (*UNTIL CORRECT IS YES*)
BEGIN
WRITELN(OUTPUT); PAGING;
WRITELN(OUTPUT, 'PLEASE CHECK THE FOLLOWING'); PAGING;
WRITELN(OUTPUT, 'FIELD NAMES FOR ERRORS:'); PAGING;
FOR COUNT := 1 TO FIELDCNT DO
BEGIN
WRITELN(OUTPUT, FIELDDID[COUNT]);
PAGING;
END;
WRITELN(OUTPUT); PAGING;
WRITELN(OUTPUT, 'ARE THE FIELD NAMES CORRECT?'); PAGING;
WRITE(OUTPUT, '->')
OKSET := ['Y', 'y', 'N', 'n'];
GETANSWER (OKSET, ANSWER);
WRITELN(OUTPUT, ANSWER); PAGING;
IF ANSWER IN ['N', 'n']
THEN BEGIN
WRITELN(OUTPUT); PAGING;
WRITELN(OUTPUT, 'ENTER THE IDS OF INCORRECT'); PAGING;
WRITELN(OUTPUT, 'FIELD NAMES SEPARATED BY COMMAS:'); PAGING;
READLN(INPUTERRORIDS)
ERRORIDS := NULL;
END
ELSE BEGIN
(* THERE ARE FIELDS TO CORRECT *)
REPEAT (*UNTIL LENGTH OF ERRORIDS = 0 *)
BEGIN
ERRORID := ERRORIDS[1]
(* GET ID OF ONE FIELD *)
IF ERRORID IN ['A'..'Z','0'..'9']
THEN BEGIN
IF ERRORID IN ['A'..'Z']
THEN FIELDNO := ORD(ERRORID) - 64
ELSE IF ERRORID IN ['0'..'9']
THEN FIELDNO := ORD(ERRORID) - 47
IF FIELDNO > FIELDCNT
THEN BEGIN
IF FIELDCNT = 0
THEN WRITELN(OUTPUT, 'NO FIELDS EXIST')
ELSE WRITELN(OUTPUT, 'NO FIELDS EXIST AFTER ',
FIELDDID[FIELDCNT]);
PAGING;
ERRORIDS := NULL;
END
ELSE BEGIN
(* LIST FIELDNAME AND ASK FOR NEW INPUT *)
WRITELN(OUTPUT, 'FIELD ', ERRORID,
',FIELDNAME[FIELDNO]);
PAGING;
WRITE(OUTPUT, 'NEW DATA:');
READLN(INPUT, INFO)
PAGING;
IF INFO <> NULL
THEN FIELDNAME[FIELDNO] := INFO;
END;
END.
DELETE(ERRORIDS,1,1); UNTIL LENGTH(ERRORIDS) = 0; END

ELSE CORRECT := YES;
WRITELN(OUTPUT): PAGING;
IF FIELDCNT <> 36 THEN BEGIN
  IF FIELDCNT = 0 THEN
    WRITELN(OUTPUT,'ARE THERE ANY FIELDS TO ADD?')
  ELSE WRITELN(OUTPUT,'ARE THERE ANY FIELDS TO ADD AFTER ', FIELDID(FIELDCNT), '?');
  PAGING;
  WRITE(OUTPUT,'->');
  OKSET := ["Y","N","y","n"];
  GETANSWER(OKSET,ANSWER);
  WRITELN(OUTPUT,ANSWER);
  IF ANSWER IN ["Y","y"] THEN BEGIN
    CORRECT := NO;
    ADDFLDNAMS;
  END;
END;
UNTIL CORRECT = YES;
END; (* FLDNAMFIX *)

PROCEDURE NAMFIX; (* ALLOWS CORRECTION OF FILE AND RECORD NAMES *)
VAR
  ANSWER: CHAR; (*USER SINGLE CHAR INPUT *)
  CORRECT: INTEGER; (*IF YES - DATA OK *)
  INFO: STRING[80]; (*INPUT INFORMATION *)
  OKSET: SETOFCHAR; (*LEGAL INPUT CHAR *)
BEGIN
  CORRECT := NO;
  REPEAT (*UNTIL CORRECT IS YES*)
    WRITELN(OUTPUT): PAGING;
    WRITELN(OUTPUT,'FILE NAME: ',FILENAME): PAGING;
    WRITELN(OUTPUT,'RECORD NAME (SINGULAR): ',RECNAMSING): PAGING;
    WRITELN(OUTPUT,'RECORD NAME (PLURAL): ',RECNAMPLUR): PAGING;
    WRITELN(OUTPUT,'ARE THESE NAMES CORRECT?'): PAGING;
    WRITE(OUTPUT,'->');
    OKSET := ["Y","y","N","n"];
    GETANSWER(OKSET,ANSWER);
    WRITELN(OUTPUT,ANSWER);
    IF ANSWER IN ["N","n"] THEN BEGIN
      WRITELN(OUTPUT);
      WRITELN(OUTPUT,'AFTER EACH PROMPT, ENTER CORRECT INPUT');
      WRITELN(OUTPUT,'"RETURN", IF INPUT IS ALREADY CORRECT');
      WRITELN(OUTPUT): PAGING;
    END;
  END; (* NAMFIX *)
WRITE(OUTPUT,'FILE NAME: ');
GETFNAME;
WRITE(OUTPUT,'RECORD NAME (SINGULAR): ');
READLN(INPUT,INFO); PAGING;
IF INFO <> NULL THEN RECNAMSING := INFO;
WRITE(OUTPUT,'RECORD NAME (PLURAL): ');
READLN(INPUT,INFO); PAGING;
IF INFO <> NULL THEN RECNAMPLUR := INFO;
END ELSE CORRECT := YES;
UNTIL CORRECT = YES;
END; (* NAMFIX *)

PROCEDURE CREATEFILE;
(* SETS UP NEW FILE ACCORDING TO INPUT. CREATE *)
(* GIVES AN OPTION TO EXIT OR CONTINUE. *)
(* CREATE PROMPTS FOR RECORD AND FIELD NAMES. *)
(* A CHECK IS MADE: IS THE REQUIRED PI-FILE *)
(* VOLUME ONLINE? ONCE THE INPUT IS CORRECT *)
(* TWO PI-FILE FILES ARE CREATED: THE HEADER *)
(* FILE AND AN EMPTY DATA FILE. *)
VAR
  ANSWER : CHAR; (* USER SINGLE CHAR INPUT *)
  IORES := INTEGER; (* I/O RESULT *)
  NODISK := INTEGER; (* FOR CALLING DISKADD *)
  OKSET := SETOFCHAR; (* LEGAL INPUT CHAR *)
  WAITCOUNT := INTEGER; (* WAIT LENGTH *)
  WHOLEFILENAME : STRING[23]; (* COMPLETE FILE NAME *)
BEGIN
(* TITLE PAGE AND EXIT OPTION *)
PAGE(OUTPUT);
LINECOUNT := 0;
GOTOXY(0,3);
WRITELN(OUTPUT,'- Create -':27);
GOTOXY(0,20);
WRITELN(OUTPUT,'Type <space> to continue':29);
WRITELN(OUTPUT,'Or <E> to exit Create':28);
OKSET := ['E', 'e', 'SPACE'];
GETANSWER(OKSET,ANSWER);
IF ANSWER = SPACE THEN BEGIN (* CREATE FILE *)
  REWRITE(DFILE,'PI-STAR: TEMP');
  CLOSE(DFILE,PURGE);
  (* CHECK FOR FILE OF SAME FNAME ALREADY EXISTING *)
  WHOLEFILENAME := CONCAT('PI-FILE:',FNAME,'.HEAD');
  (**I**)  RESET(DFILE,WHOLEFILENAME);
  IORES := IORESFILE;
  (**I**)  IF IORES = 0
THEN BEGIN
    WRITELN(OUTPUT,'FILE ',FNAME,' ALREADY EXISTS');
    WAITCOUNT := 40;
    REPEAT
        WAITCOUNT := WAITCOUNT - 1;
        NOTE(0,15);
    UNTIL WAITCOUNT = 0;
    CLOSE(DFILE,NORMAL);
    END ELSE BEGIN
    PAGE(OUTPUT);
    LINECOUNT := 0;
    (* GET INFO *)
    WRITELN(OUTPUT,'WHAT IS THE NAME OF A SINGLE RECORD?'); PAGING;
    READDLN(INPUT,RECNAMSING); PAGING;
    WRITELN(OUTPUT,'WHAT IS ITS PLURAL?'); PAGING;
    READDLN(INPUT,RECNAMPLUR); PAGING;
    (* ALLOW CORRECTIONS ON FILE AND RECORD NAMES *)
    NAMFIX;
    (* INPUT FIELD NAMES *)
    FIECNT := 0;
    ADDFLDNAM;
    (* ALLOW CORRECTIONS ON INPUT INFORMATION *)
    FLDNAMFIX;
    (* CREATE HEADER AND DATA FILES *)
    WRITELN(OUTPUT);
    WRITELN(OUTPUT,'PLEASE WAIT -'); PAGING;
    WRITELN(OUTPUT,'FILE BEING CREATED'); PAGING;
    DISKADD(NODISK); (* TEST FOR PI-FILE ONLINE AND CREATE FILE *)
    END;
END; (*CREATEFILE*)
END; (*CREATE*)

BEGIN
    (* INITIALI ZATION CODE *)
END.
(*  PRINT *)

UNIT TRANSFORM:
(* CONVERTS THE QUERY TO POSTFIX NOTATION *)

 INTERFACE

USES APPLESTUFF, GLOBAL, SUPPORT, EDIT;

 PROCEDURE TRANS;

 IMPLEMENTATION

 TYPE
 GENERICTOKENS = (TGID, (* AN IDENTIFIER *)
 TOREL, (* A RELATIONAL OPERATOR *)
 TOLOG, (* A LOGICAL OPERATOR *)
 TOSTR, (* A SEARCH STRING *)
 TOLPAR, (* A LEFT PARENTHESIS *)
 TORPAR, (* A RIGHT PARENTHESIS *)
 TODELM, (* TORY DELIMITER *)
 TONOT); (* NOT OPERATOR *)

 STATE = (ST0, ST1, ST2, ST3, (* NORMAL STATE *)
 ERR, ACPT); (* ERROR, ACCEPT *)

 VAR
 CHARACTER : CHAR; (* A SINGLE CHARACTER OF QUERY *)
 COUNT : INTEGER; (* TEMPORARY ARRAY INDEX *)
 LTRIDS : PACKED ARRAY[65..90] OF IDA..IDZ; (* LETTER IDENTIFIERS *)
 MSG : STRING[76]; (* ERROR MSG PASSED TO CORRECT *)
 NUMIDS : PACKED ARRAY[48..57] OF IDO..ID9; (* NUMBER IDENTIFIERS *)
 STATEARRAY : PACKED ARRAY[ST0..ST3, GENERICTOKENS] OF STATE; (* FINITE STATE MATRIX *)
 STINDEX : ST0..ST3; (* TEMP INDEX INTO STATE MATRIX *)
 STRCNT : STR1..STR15; (* SEARCH STRING IDENTIFIERS *)
 STRFLAG : INTEGER; (* IF YES - MORE STRINGS ALLOWED *)
 QCOUNT : INTEGER; (* CURRENT ELEMENT OF QUERY *)
 TCOUNT : INTEGER; (* NUMB TOKENS IN TOUERY & TORY *)
 TEMP : IDA..ID9; (* TEMPORARY IDENTIFIER *)
 TKNFND : PACKED ARRAY[TKID, TONOT] OF STRING[19]; (* FOUND TOKEN *)
 TDFLAG : INTEGER; (* IF YES - TRANSLATION OK *)
 TOINDEX : GENERICTOKENS; (* TEMP INDEX INTO STATE MATRIX *)
 TORY : PACKED ARRAY[1..90] OF GENERICTOKENS; (* TOKENIZED QUERY FOR SNTX CHK *)
 TOQUERY : PACKED ARRAY[1..90] OF TOKENS; (* TOKENIZED QUERY FOR PFX Conv *)

 PROCEDURE IDENTIFIER:
 (* PLACES THE PROPER ID TOKEN INTO TOQUERY *)
VAR
  INDEX : INTEGER; (* INDEX INTO ARRAY *);
BEGIN
  IF CHARACTER <> '#' THEN BEGIN
    IF CHARACTER IN ['A'..'Z'] THEN BEGIN (* IDA..IDZ *)
      INDEX := ORD(CHARACTER);
      TOQUERY[TCOUNT] := LTRIDS[INDEX];
      END
    ELSE BEGIN (* ID0..ID9 *)
      INDEX := ORD(CHARACTER);
      TOQUERY[TCOUNT] := NUMIDS[INDEX];
      END
    END
  END
  ELSE TOQUERY[TCOUNT] := IDAND; (* ID *)
  TCOUNT := TCOUNT + 1;
END; (* IDENTIFIER *)

PROCEDURE LOGICALOP;
  (* LOGICALOP DETERMINES IF A COMPLETE LOGICAL OP *)
  (* (/AND/,/OR/,/NOT/) HAS BEEN FOUND. IF IT HAS, *)
  (* THEN THE PROPER TOKEN IS PLACED IN TOQUERY. *)
VAR
  COUNT : INTEGER; (* COUNTER *)
  TMPCHARS : STRING[4]; (* CHARACTERS OF QUERY *)
BEGIN
  (* GET THREE CHARACTERS FROM QUERY *)
  TMPCHARS := ' ';
  COUNT := 1;
  WHILE (OCOUNT < ORYCNT) AND (COUNT <= 3) DO BEGIN
    OCOUNT := OCOUNT + 1;
    TMPCHARS[OCOUNT] := QUERY[OCOUNT];
    COUNT := COUNT + 1;
  END;
  IF TMPCHARS <> 'OR/' THEN BEGIN
    (* GET NEXT CHARACTER FROM QUERY *)
    TMPCHARS := CONCAT(TMPCHARS, SPACE);
    IF OCOUNT < ORYCNT THEN BEGIN
      OCOUNT := OCOUNT + 1;
      TMPCHARS[4] := QUERY[OCOUNT];
    END;
    IF TMPCHARS <> 'AND/' THEN IF TMPCHARS <> 'NOT/' THEN BEGIN
      TOFLAG := NO;
      MSG := 'ILLEGAL LOGICAL OPERATOR';
    END;
  END;
END; (* IDENTIFIER */
END
ELSE TQUERY(TCOUNT) := TNOT
ELSE TQUERY(TCOUNT) := TAND;
END
ELSE TQUERY(TCOUNT) := TOR;
IF TOFLAG = YES
THEN BEGIN
IF TQUERY(TCOUNT) <> TNOT
THEN TORYETCOUNT := TOLOG
ELSE TORYETCOUNT := TNOT;
TCOUNT := TCOUNT + 1;
END;
END; (* LOGICALOP *)

PROCEDURE RELATIONALOP;
(* RELATIONAL OP DETERMINES WHAT RELATIONAL OPERATOR *)
(* HAS BEEN FOUND, AND ADD PROPER TOKEN TO TQUERY. *)
VAR
    OP : STRING[2]; (* OPERATOR *)
BEGIN
    IF CHARACTER IN [',','"']
    THEN IF ((QUERY[COUNT+1] = '=' OR (QUERY[COUNT+1] = ' >'))
        AND (COUNT < QRYCNT)
    THEN BEGIN
        COUNT := COUNT + 1;
        OP[1] := ' ';
        OP[2] := QUERY[COUNT];
        IF OP = '<='
        THEN TQUERY(TCOUNT) := LE
        ELSE IF OP = '>='
        THEN TQUERY(TCOUNT) := GE
        ELSE IF OP = '<>'
        THEN TQUERY(TCOUNT) := NE
        ELSE BEGIN
            (* ERROR *)
            TQFLAG := NO;
            MSG := 'ILLEGAL RELATIONAL OPERATOR: >>';
        END; (* CASE *)
    END
ELSE IF CHARACTER = '
    THEN TQUERY(TCOUNT) := LT
ELSE TQUERY(TCOUNT) := GT
ELSE TQUERY(TCOUNT) := EQ;
IF TOFLAG = YES
THEN BEGIN
    TORYETCOUNT := TOREL;
    TCOUNT := TCOUNT + 1;
END;
END; (* RELATIONALOP *)

PROCEDURE SRCHSTRING;
(* PUTS THE SEARCH STRING IN SRCHSTR ARRAY, AND PUTS *)
(* AN INDEX INTO THE ARRAY INTO TQUERY AS A TOKEN. *)

VAR
  CHARCOUNT : INTEGER; (* NUMBER OF CHARACTERS IN STR *)
  DBLUTE : INTEGER; (* INDICATES STRING DELIMITER *)
  DONE : INTEGER; (* IF YES- STRING COMPLETE OR ERR *)
  TMPSTR : STRING[11]; (* CHARACTER *)
BEGIN
  IF STRFLAG = YES
    THEN BEGIN
      SRCHSTR[STRCNT] := NULL;
      TMPSTR := SPACE;
      DONE := NO;
      CHARCOUNT := 0;
      IF CHARACTER = '"'
        THEN DBLUTE := YES (* DELIMITER IS QUOTE *)
      ELSE DBLUTE := NO; (* DELIMITER IS APOSTROPHE *)
      WHILE (QCOUNT < QRYCNT) AND (DONE = NO) DO
        BEGIN
          IF QCOUNT < QRYCNT
            THEN BEGIN
              QCOUNT := QCOUNT + 1;
              CHARACTER := QUERY[QCOUNT];
            END;
          IF NOT((CHARACTER = '"') AND (DBLUTE = YES))
            THEN IF NOT((CHARACTER = '''') AND (DBLUTE = NO))
              BEGIN (* ADD CHAR TO STRING *)
                CHARCOUNT := CHARCOUNT + 1;
                IF CHARCOUNT > 255
                  THEN BEGIN
                    DONE := YES;
                    TOFLAG := NO;
                    MSG := 'STRING TOO LONG';
                  END;
                ELSE BEGIN
                  TMPSTR[1] := CHARACTER;
                  IF TMPSTR[1] = DELIMITER
                    THEN SRCHSTR[STRCNT] :=
                              CONCAT(SRCHSTR[STRCNT], SPACE)
                  ELSE SRCHSTR[STRCNT] :=
                              CONCAT(SRCHSTR[STRCNT], TMPSTR);
                END
              END;
            ELSE DONE := YES
            ELSE DONE := YES;
          END; (* WHILE *)
    END;
  END; (* IF *)
END; (* IF *)
TQUERY[TCOUNT] := STRCNT;
TQRY[TCOUNT] := TOSTR;
TCOUNT := TCOUNT + 1;
IF STRCNT <> STR15
THEN STRCNT := SUCC(STRCNT)
ELSE STRFLAG := NO; (* NO MORE SRCHSTRS ALLOWED *)
END;
END ELSE BEGIN
TOFLAG := NO;
MSG := 'TOO MANY SEARCH STRINGS';
END;
END; (* SRCHSTRING *)

PROCEDURE PARENTHESIS;
(* PUTS THE PROPER TOKEN INTO TQUERY IF A LEFT *)
(* OR RIGHT PARENTHESIS IS FOUND. *)
BEGIN
IF CHARACTER = '('
THEN BEGIN
TQUERY[TCOUNT] := LPAR;
TQRY[TCOUNT] := TLAPAR;
END
ELSE BEGIN
TQUERY[TCOUNT] := RPAR;
TQRY[TCOUNT] := TRPAR;
END;
TCOUNT := TCOUNT + 1;
END; (* PARENTHESIS *)

PROCEDURE TOKENIZE;
(* TOKENIZE TAKES THE ORIGINAL QUERY AND BUILDS A NEW QUERY *)
(* BY REPLACING THE ORIGINAL QUERY ELEMENTS WITH TOKENS. *)
BEGIN
TOFLAG := YES;
STRFLAG := YES;
OCOUNT := O; (* POINTS TO CURRENT ELEMENT OF QUERY *)
TCOUNT := 1; (* NUMBER OF CHARACTERS IN TOUERY *)
STRCNT := STR1;
(* GET FIRST CHARACTER OF QUERY *)
CHARACTER := SPACE;
WHILE (CHARACTER IN [DELTMS]) AND (OCOUNT <= QRYCNT) DO BEGIN
OCOUNT := OCOUNT + 1;
IF OCOUNT <= QRYCNT
THEN CHARACTER := QUERY[OCOUNT];
END;
WHILE (OCOUNT <= QRYCNT) AND (TOFLAG = YES) DO BEGIN
IF CHARACTER IN ['#', 'A'..'Z', '0'..'9', '/', ',', '(', ')', '.']
THEN CASE CHARACTER OF
BEGIN

(* CASE *)
ELSE BEGIN (* ERROR IN QUERY *)
TQFLAG := NO;
MSG := 'ILLEGAL CHARACTER';
END;

(* GET NEXT CHARACTER OF QUERY *)
CHARACTER := SPACE;
IF OCOUNT < ORYCNT
THEN
  WHILE ((CHARACTER IN (DELIMITER, SPACE)) AND (OCOUNT <= ORYCNT)) DO
    BEGIN
      OCOUNT := OCOUNT + 1;
      IF OCOUNT <= ORYCNT
      THEN CHARACTER := QUERY[OCOUNT];
    END;
  ELSE OCOUNT := OCOUNT + 1;
END; (* WHILE *)
IF TQFLAG = YES
THEN BEGIN
  TOQUERY[TCOUNT] := DELM;
  TORY[TCOUNT] := TODELM;
END;
END; (* TOKENIZE *)

PROCEDURE SYNTAXCHECK;
(* SYNTAXCHECK TAKES THE GENERIC QUERY TORY AND CHECKS IT AGAINST A FINITE STATE MATRIX WHICH REPRESENTS A LEGAL QUERY.*)
VAR
CURRSTATE : ST0..ST3; (* CURRENT STATE *)
LPARCNT : INTEGER; (* LEFT PAREN COUNTER *)
NEWSTATE : STATE; (* RTNED FROM STATEARRAY *)
RPARCNT : INTEGER; (* RIGHT PAREN COUNTER *)
TORYCNT : INTEGER; (* POINTS TO CURR ELEMENT - TORY *)
TKN : GENERICTOKENS; (* ONE TOKEN FROM TORY *)
BEGIN
LPARCNT := 0; (* LEFT PARENTHESIS COUNT *)
RPARCNT := 0; (* RIGHT PARENTHESIS COUNT *)
TORYCNT := 1;
CURRSTATE := ST0; (* BEGIN IN STATE ZERO *)
TQFLAG := YES; (* SYNTAX IS OK *)
TKN := T0ID;
WHILE (TKN <> TODELM) AND (TQFLAG = YES) DO
  BEGIN
    (* PROCESS QUERY CHARACTERS *)
    THEN BEGIN
      (* IDENTIFIER *)
      TQFLAG := YES;
      MSG := 'IDENTIFIER';
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* LOGICALOP *)
      TQFLAG := YES;
      MSG := 'LOGICALOP';
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* RELATIONALOP *)
      TQFLAG := YES;
      MSG := 'RELATIONALOP';
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* SRCHSTRING *)
      TQFLAG := YES;
      MSG := 'SRCHSTRING';
    END;
    IF CHARACTER IN ('(', ')', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* PARENTHESIS *)
      TQFLAG := YES;
      MSG := 'PARENTHESIS';
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* CASE *)
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* ERROR IN QUERY *)
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* GET NEXT CHARACTER OF QUERY *)
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* WHILE *)
    END;
    IF CHARACTER IN ('=', '>', '<', '!', 'R', 'H', 'S', ')', '(', '!', 'I', '5', '6', '0')
    THEN BEGIN
      (* TOKENIZE *)
    END;
  END;
END; (* PROCEDURE SYNTAXCHECK *)

END;
TKN := TORYCNT;
TORYCNT := TORYCNT + 1;
NEWSTATE := STATEARRAY[CURRSTATE, TKN];
IF NOT(NEWSTATE IN [ST0..ST3])
THEN BEGIN
  IF NEWSTATE = ERR
  THEN IF TKN = TODELM
  THEN BEGIN
    MSG := 'UNEXPECTED END OF QUERY';
    TOFLAG := NO;
  END
  ELSE BEGIN (# SYNTAX ERROR *)
    TOFLAG := NO;
    MSG := CONCAT('UNEXPECTED ', TKNFND[TKN]);
  END
  ELSE BEGIN (* ACCEPT STATE *)
    IF LPARCNT < RPARCNT
    THEN BEGIN
      TOFLAG := NO;
      MSG := 'MISSING ONE OR MORE RIGHT PARENTHESES';
    END
    ELSE BEGIN (* OK TOKEN *)
      IF TKN = TOLPAR
      THEN BEGIN
        LPARCNT := LPARCNT + 1;
        IF LPARCNT > RPARCNT
        THEN BEGIN
          TOFLAG := NO;
          MSG := 'MISSING LEFT PARENTHESES';
        END
        ELSE IF TKN = TOLPAR
        THEN LPARCNT := LPARCNT + 1;
        CURRSTATE := NEWSTATE;
      END
    END
  END
END; (* SYNTAX CHECK *)

PROCEDURE POSTFIX;
  (* TRANSLATES THE TOKENIZED QUERY INTO POSTFIX (POLISH) NOTATION. *)
  (* STACK CONDITION *)
  (* TOKEN JUST POPPED FROM STACK *)
  (* INDEX INTO QUERY *)
  (* PACKED ARRAY OF..90. OF TOKENS *)
  (* OPERATOR STACK *)
  (* FIRST EMPTY STACK POSITION *)
  (* ONE ELEMENT OF QUERY *)
  (* INDEX INTO QUERY *)
  VAR
    EMPTY : INTEGER; (* STACK CONDITION *)
    PTKN : TOKENS; (* TOKEN JUST POPPED FROM STACK *)
    FORYCNT : INTEGER; (* INDEX INTO QUERY *)
    STACK : PACKED ARRAY[0..90] OF TOKENS; (* OPERATOR STACK *)
    STACKFTR : INTEGER; (* FIRST EMPTY STACK POSITION *)
    TKN : TOKENS; (* ONE ELEMENT OF QUERY *)
    TORYCNT : INTEGER; (* INDEX INTO QUERY *)

PROCEDURE PUSH;

(* PUSHES A TKN ONTO THE STACK *)
BEGIN
  IF STACKPTR <= 90 THEN BEGIN
    STACK[STACKPTR] := TKN;
    STACKPTR := STACKPTR + 1;
    EMPTY := NO;
  END ELSE BEGIN
    POPFLAG := NO;
    MSG := 'QUERY TOO COMPLEX';
  END;
END; (* PUSH *)

PROCEDURE POP;

(* REMOVES A TKN FROM THE STACK *)
BEGIN
  STACKPTR := STACKPTR - 1;
  PTKN := STACK[STACKPTR];
  IF STACKPTR = 1 THEN EMPTY := YES;
END; (* POP *)

BEGIN
  POPFLAG := YES;
  STACKPTR := 1; (* STACK EMPTY *)
  STACK[0] := DELM;
  EMPTY := YES;
  PQRYCNT := 1;
  TQRYCNT := 1;
  (* GET FIRST TOKEN FROM QUERY *)
  TKN := TOUERY[TQRYCNT];
  TQRYCNT := TQRYCNT + 1;
  PQUERY := 1;
  PQRYCNT := PQRYCNT + 1;
  (* GET FIRST TOKEN FROM TOUERY *)
  TKN := TOUERY[PQUERY];
  PQUERY := PQUERY + 1;
  PQUERY := PQUERY + 1;
  IF TKN = ID9 THEN PUSH (* TKN *)
  ELSE BEGIN
    POP;
    PQUERY := PQUERY - 1;
    PQUERY := PQUERY + 1;
    PQUERY := PQUERY + 1;
  END;
END; (* GET TOKEN *)
PUSH; (* TKN *)
END
ELSE BEGIN
   WHILE STACK[STACKPTR-1] <> LPAR DO
      BEGIN
         POP; (* P'TKN *)
         POQUERY[PORYCNT] := TKN;
         PORYCNT := PORYCNT + 1;
      END;
      POP; (* REMOVE LPAR *)
   END
ELSE PUSH (* LPAR *)
ELSE BEGIN (* ADD OPERAND TO POQUERY *)
   POQUERY[PORYCNT] := TKN;
   PORYCNT := PORYCNT + 1;
END;
(* GET TOKEN *)
TKN := POQUERY[TORYCNT];
TORYCNT := TORYCNT + 1;
END; (* WHILE *)
IF POFLAG = YES
   THEN BEGIN
      WHILE EMPTY = NO DO
         BEGIN
            POP; (* P'TKN *)
            POQUERY[PORYCNT] := TKN;
            PORYCNT := PORYCNT + 1;
         END;
         POQUERY[PORYCNT] := DELM;
      END;
   END; (* POSTFIX *)

PROCEDURE TRANS;
(* CALLS THE ROUTINES NEEDED TO CREATE A QUERY WHICH IS *)
(* UNDERSTOOD BY THE EXPRESSION EVALUATOR ROUTINES. *)
(* ALLOWS CORRECTION OF ORIGINAL QUERY IF ERROR IS FOUND *)
VAR
   ANSWER : CHAR; (* USER RESPONSE *)
   ERROR : INTEGER; (* IF YES - ERROR IN QUERY *)
   OKSET : SETOFCHAR; (* LEGAL INPUT CHAR *)
   USREXIT : INTEGER; (* IF YES - USER SAYS EXIT *)
BEGIN
   POFLAG := NO;
   ERROR := YES;
   USREXIT := NO;
   REPEAT
      TOKENIZE;
      IF TOFLAG = YES (* NO ERRORS FOUND *)
         THEN BEGIN
            SYNTAXCHECK;
            IF TOFLAG = YES (* NO ERRORS FOUND *)
               THEN BEGIN


POSTFIX;
  IF POFLAG = YES (* NO ERRORS FOUND *)
    THEN ERROR := NO;
END;

IF ERROR = YES THEN BEGIN (* MAKE CORRECTIONS *)
  EDITOR(QUERY,ORYCNT,MSG);
  WRITELN(OUTPUT); PAGING;
  WRITELN(OUTPUT); PAGING;
  WRITELN(OUTPUT,'EDIT COMPLETE'); PAGING;
  WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE SEARCH'); PAGING;
  WRITELN(OUTPUT,'<E> TO RETURN TO "INPUT QUERY" MENU.'); PAGING;
  USREXIT := (SPACE,'E','e');
  GETANSWER(OKSET,ANSWER);
  IF ANSWER IN ['E','e'] THEN USREXIT := YES;
END;
UNTIL (ERROR = NO) OR (USREXIT = YES);
END; (* TRANS *)

BEGIN (* INITIALIZE SECTION *)
(* INITIALIZE IDENTIFIERS *)
  TEMP := IDA;
  FOR COUNT := 65 TO 90 DO BEGIN
    LTRIDSECOUNTJ := TEMP;
    IF TEMP <> IDZ THEN TEMP := SUCC(TEMP);
  END;
  TEMP := IDO;
  FOR COUNT := 48 TO 57 DO BEGIN
    NUMIDSECOUNT3 := TEMP;
    IF TEMP <> ID9 THEN TEMP := SUCC(TEMP);
  END;
(* INITIALIZE TKNFND *)
  TKNFND[TOID] := 'IDENTIFIER';
  TKNFND[TOREL] := 'RELATIONAL OPERATOR';
  TKNFND[TOLOG] := 'LOGICAL OPERATOR';
  TKNFND[TOSTR] := 'SEARCH STRING';
  TKNFND[TOLPAR] := '(';
  TKNFND[TORPAR] := ')';
  TKNFND[TONOT] := 'LOGICAL OPERATOR';
(* INITIALIZE FINITE STATE MATRIX *)
  FOR STINDEX := TOID TO TONOT DO BEGIN
    FOR STINDEX := STO TO ST3 DO
      STATEARRAY[STINDEX,STINDEX] := ERR;
      STATEARRAY[ST0,TOID] := ST1;
      STATEARRAY[ST1,TOREL] := ST2;
      STATEARRAY[ST2,TOLOG] := ST0;
STATEARRAY ST2, TOSTR := ST3;
STATEARRAY(STO, TOLPARJ) := ST0;
STATEARRAY(ST3, TOLPARJ) := ST3;
STATEARRAY(T3, TODELMJ) := ACFT;
STATEARRAY(ST0, TONOT) := ST0;
END.

(*$S+*)
(*$L PRINTER:*)
UNIT ADDRESS:
(* ADDREC ALLOWS THE INPUTTING OF NEW RECORDS *)
(* TO AN EXISTING FILE, IT ALSO TAKES A BLANK *)
(* FORMATTED PI-FILE DISK AND SETS UP THE *)
(* PI-STAR HEADER AND DATA FILES ON THAT DISK.*)

INTERFACE
USES APPLESTUFF, GLOBAL, SUPPORT, EDIT;
PROCEDURE ADD;
IMPLEMENTATION
VAR
  NUMBCHAR : INTEGER; (* NUMBER OF CHARS IN FIELD *)
PROCEDURE CORRECTREC (COUNT: INTEGER);
(* ALLOWS USER TO CORRECT THE CURRENT FIELD *)
VAR
  ANSWER : CHAR; (* USER SINGLE CHAR INPUT *)
  CORRECT : INTEGER; (* IF YES - DATA OK *)
  MESSAGE : STRING[76]; (* PASSED TO EDITOR *)
  OKSET := SETOFCHAR; (* LEGAL SET OF CHARS *)
BEGIN
  CORRECT := NO;
  REPEAT (* UNTIL CORRECT IS YES *)
    WRITELN(OUTPUT); PAGING;
    WRITELN(OUTPUT, 'FIELD ', FIELDNAME[COUNT], ':'); PAGING;
    WRITELN(OUTPUT); PAGING;
    WRITELN(OUTPUT); PAGING;
    WRITELN(OUTPUT, 'IS THE FIELD CORRECT?'); PAGING;
    WRITE(OUTPUT, '"');
    OKSET := ['Y', 'y', 'N', 'n'];
    GETANSWER(OKSET, ANSWER);
    WRITELN(OUTPUT, ANSWER); PAGING;
    WRITELN(OUTPUT); PAGING;
    IF ANSWER IN ['N', 'n'] THEN BEGIN
      MESSAGE := CONCAT('EDIT ', FIELDNAME[COUNT], ' OF ', RECNAMSING);
      EDITOR(FIELD, NUMBCHAR, MESSAGE);
    END
      ELSE CORRECT := YES;
      UNTIL CORRECT = YES;
END; (* CORRECTREC *)

PROCEDURE INPUTREC;
(*) INPUT THE FIELDS OF THE RECORDS BEING *)
(*) ADDED TO THE FILE. *)

VAR
COUNT   : INTEGER; (*LOOP COUNTER *)
COUNTA  : INTEGER; (*LOOP COUNTER *)
MESSAGE : STRING[76]; (*PASSED TO EDITOR *)
TSTR    : STRING[11]; (*TEMP STRING *)
WHOLEFILENAME : STRING[13]; (*FILE NAME *)

BEGIN
(* ADD # (MARKED/NOT MARKED) FIELD TO TFILE *)
(* N = NOT MARKED *)
REWRITE(TFILE,'PI-STAR:FLD,PND');
WRITE(TFILE,'N');
CLOSE(TFILE,LOCK);
(* INPUT AND CORRECT ONE FIELD AT A TIME UNTIL ALL ARE INPUT *)
COUNT := 0;
REPEAT
COUNT := COUNT +1;
NUMBCHAR := 0;
MESSAGE := CONCAT('INPUT ',FIELDNAME[COUNT], ' OF ',RECNAMSING);
EDITOR(FIELD,NUMBCHAR,MESSAGE);
CORRECTREC(COUNT);
(* WRITE THE FIELD TO TFILE *)
COUNTA := 0;
TSTR := SPACE;
TSTR[1] := FIELDID[COUNT];
WHOLEFILENAME := CONCAT('PI-STAR:FLD, ',TSTR);
REWRITE(TFILE,WHOLEFILENAME);
IF NUMBCHAR <> 0
  THEN BEGIN
    REPEAT
      COUNTA := COUNTA +1;
      WRITE(TFILE,FIELD[COUNTA]);
    UNTIL COUNTA = NUMBCHAR;
  END;
  CLOSE(TFILE,LOCK);
UNTIL COUNT = FIELDCNT;
END; (* INPUTREC *)

PROCEDURE ADD;
(* ADD PREPARES FOR ADDING RECORDS TO AN EXISTING *)
(* FILE OR SETS UP THE DATA FILES ON AN EMPTY *)
(* PI-FILE DISK. *)

VAR
ANSWER : CHAR; (* USER SINGLE CHAR INPUT *)
COUNT   : INTEGER; (* LOOP COUNTER *)
FOPEN   : INTEGER; (* IF YES- OPEN FILE *)
IRES    : INTEGER; (* I/O RESULT *)
MORERECS : INTEGER; (* IF YES- MORE RECORDS *)
NO_DISK : INTEGER; (* IF YES- NO PI-FILE DISK *)
OKSET   : SETOFCHAR; (* LEGAL INPUT SET *)
TSTR    : STRING[11]; (* TEMP STRING *)
WAITCOUNT : INTEGER; (* COUNT TO WAIT *)
WHOLEFILENAME : STRING(23); (* COMPLETE FILENAME *)
BEGIN
(* TITLE PAGE *)
PAGE(OUTPUT); LINECOUNT := 0;
GOTOXY(0,3);
WRITELN(OUTPUT,'- A D D -';24);
GOTOXY(0,20);
WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE';29);
WRITELN(OUTPUT,'<E> TO EXIT ADD';25);
OKSET := ['E',SPACEJ);
GETANSWER(OKSETANSWER);
PAGE(OUTPUT); LINECOUNT := 0;
IF ANSWER = SPACE THEN BEGIN (* ADD RECORD *)
(* GET FILE HEADER *)
PAGERECORD := 0;
GOTOXY(0,3);
WRITELN(OUTPUT,'TO ADD RECORD(S), TYPE <R>'); PAGING;
WRITELN(OUTPUT,'TO ADD DISK, TYPE <D>'); PAGING;
OKSET := ['D',SPACEJ);
GETANSWER(OKSETANSWER);
WRITELN(OUTPUT,ANSWER); PAGING;
WHOLEFILENAME := CONCAT('PI-FILE:',FNAME,HEAD);
(*$I-*)
RESET FILEL, WHOLEFILENAME);
(*$I+*)
IF IORES = C0 THEN BEGIN
CLOSE FILEL, NORMAL);
(* CLOSE TO FORCE APPLE TO GET *)
(* DIRECTORY OF NEW DISK. *)
REWRITE(DFILE,'PI-STAR:TEMP');
CLOSE(DFILE,Purge);
WRITELN(OUTPUT,'PUT IN BLANK PI-FILE DISK'); PAGING;
WRITELN(OUTPUT,'AND TYPE <SPACE> WHEN READY'); PAGING;
OKSET := [SPACEJ);
GETANSWER(OKSETANSWER);
END;
UNTIL IORES <> 0;
DISKADD(ODISK);
END
ELSE BEGIN
FOPEN := YES;
MORERECS := YES;

REPEAT (* UNTIL NO MORE RECORDS *)
  INPUTREC;
  WRITELN(OUTPUT,"RECORD READY TO BE WRITTEN TO FILE");
  PAGING;
  WRITELN(OUTPUT,"TYPE <SPACE> TO CONTINUE");
  PAGING;
  WRITELN(OUTPUT,"<E> TO ABORT ADD");
  PAGING;
  OKSET := ['E', 'n', 'SPACE'];
  GETANSWER(OKSET, ANSWER);
  IF ANSWER = SPACE
    THEN WRITEFILE(FOPEN);
    WRITELN(OUTPUT); PAGING;
    WRITE(OUTPUT,"ADD ANOTHER RECORD?" ); PAGING;
    WRITE(OUTPUT,", ->");
    OKSET := ['Y', 'y', 'N', 'n'];
    GETANSWER(OKSET, ANSWER);
    WRITELN(OUTPUT,ANSWER); PAGING;
    WRITELN(OUTPUT); PAGING;
    IF ANSWER IN ['N', 'n']
      THEN MORERECS := NO;
    UNTIL MORERECS = NO;

(* GET RID OF TEMPORARY FILES *)
RESET(TFILE,'PI-STAR:FLD.PND');
CLOSE(TFILE,PURGE);
FOR COUNT := 1 TO FIELDCNT DO
  BEGIN
    TSTR := SPACE;
    TSTR[] := FIELDID(COUNT);
    WHOLEFILENAME := CONCAT('PI-STAR:FLD.',TSTR);
    RESET(TFILE,WHOLEFILENAME);
    CLOSE(TFILE,PURGE);
  END;
  IF FOPEN <> YES
    THEN CLOSE(DFILE,NORMAL);
END;

END; (* ADD RECORD *)
END; (* ADD *)

BEGIN
(* INITIALIZATION SECTION *)
END.
UNIT DRYINPUT;
(* INPUT QUERY *)

INTERFACE
USES APPLESTUFF, GLOBAL, SUPPORT, EDIT, TRANSLATE;

PROCEDURE INPUTQUERY;

IMPLEMENTATION

PROCEDURE PRTORY;
(* THIS PROCEDURE GENERATES A HARDCOPY LISTING OF THE CURRENT QUERY *)
VAR
  ANSWER : CHAR; (* USER INPUT *)
  CHARCOUNT : INTEGER; (* COUNT OF CHARACTERS ON LINE *)
  COUNT : INTEGER; (* LOOP COUNTER *)
  OKSET : SETOFCHAR; (* OK USER INPUT *)
BEGIN
  IF PRTER = YES
  THEN BEGIN
    CHARCOUNT := 0;
    FOR COUNT := 1 TO ORYCNT DO
      BEGIN
        IF QUERY[COUNT] = DELIMITER
        THEN BEGIN
          WRITELN(PRINTER);
          CHARCOUNT := 0;
        END
        ELSE BEGIN
          WRITE(PRINTER, QUERY[COUNT]);
          CHARCOUNT := CHARCOUNT + 1;
        END;
        IF (CHARCOUNT = 80) OR (COUNT = ORYCNT)
        THEN (* GENERATE CR FOR EOLN OR EOD *)
          IF QUERY[COUNT] <> DELIMITER
          THEN WRITELN(PRINTER);
      END;
    WRITELN(PRINTER);
    WRITELN(PRINTER);
  END
  ELSE BEGIN
    WRITELN(OUTPUT, 'PRINTER NOT ON-LINE');
    WRITE(OUTPUT, 'TYPE <SPACE> TO CONTINUE');
    OKSET := (SPACE);
    GETANSWER(OKSET, ANSWER);
  END;
END; (* PRTORY *)
PROCEDURE DSPLYNAM;
(* GENERATES A LIST OF THE FIELD NAMES AND IDENTIFIERS ON THE *)
(* SCREEN. *)
VAR
ANSWER : CHAR; (* USER RESPONSE *)
COUNT : INTEGER; (* LOOP COUNTER *)
OKSET : SETOFCHAR; (* OK USER INPUT *)
BEGIN
PAGE(OUTPUT); LINECOUNT := 0;
WRITELN(OUTPUT, 'FILE: ',FILENAME); PAGING;
WRITELN(OUTPUT); PAGING;
WRITELN(OUTPUT, 'RECORD: ',RECNAMSING, '('RECNAMPLUR,')'); PAGING;
WRITELN(OUTPUT); PAGING;
FOR COUNT := 1 TO FIELDCNT DO
BEGIN
WRITELN(OUTPUT,FIELDIDECOUNTJ, ' ',FIELDNAMICOUNTJ);
PAGING;
END;
END; (* DSPLYNAM *)

PROCEDURE PRTNAM;
(* Generates a hardcopy listing of the file and record names and *)
(* each fieldname and identifier. *)
VAR
ANSWER : CHAR; (* USER INPUT *)
COUNT : INTEGER; (* LOOP COUNTER *)
OKSET : SETOFCHAR; (* OK USER INPUT *)
BEGIN
IF PRTER = YES THEN BEGIN
WRITELN(PRINTER, 'FILE: ',FILENAME);
WRITELN(PRINTER);
WRITELN(PRINTER, 'RECORD: ',RECNAMSING, '('RECNAMPLUR,')');
WRITELN(PRINTER);
FOR COUNT := 1 TO FIELDCNT DO
BEGIN
WRITELN(PRINTER,FIELDIDECOUNTJ, ' ',FIELDNAMICOUNTJ);
WRITELN(PRINTER);
WRITELN(PRINTER);
END;
END ELSE BEGIN
WRITELN(OUTPUT,'PRINTER NOT ON-LINE');
WRITE(OUTPUT, 'TYPE <SPACE> TO CONTINUE');
OKSET := (SPACE);
GETANSWER(OKSET,ANSWER);
END;
END; (* PRTNAM *)
PROCEDURE INPUTQUERY;
(* ALLOWS A QUERY TO BE INPUT AND TRANSLATED. A COPY *)
(* OF THE QUERY CAN BE PRINTED. A LIST OF FIELD *)
(* NAMES AND IDENTIFIERS CAN ALSO BE DISPLAYED OR *)
(* PRINTED. *)
VAR
  ANSWER : CHAR; (* USER INPUT *)
  MSG : STRING[80]; (* PROMPT MESSAGE *)
  OKSET : SETOFCHAR; (* LEGAL USER INPUT *)
BEGIN
REPEAT
  PAGE(OUTPUT); LINECOUNT := 0;
  GOTOXY(0, 5);
  WRITELN(OUTPUT, 'INPUT QUERY');
  GOTOXY(0, 10);
  WRITELN(OUTPUT, 'SELECT: ');
  WRITELN(OUTPUT, 'I)NPUT QUERY';
  WRITELN(OUTPUT, 'S)EARCH MENU');
  WRITELN(OUTPUT, 'P)RINT QUERY');
  WRITELN(OUTPUT, 'D)ISPLAY FIELD NAMES AND IDS');
  WRITELN(OUTPUT, 'O)UTPUT FIELD NAMES AND IDS');
  WRITELN(OUTPUT, 'E)XIT');
  WRITELN(OUTPUT);
  WRITE(OUTPUT, '-->');
  GETANSWER(OKSET, ANSWER);
  CASE ANSWER OF
    'I', 'I' : BEGIN (* INPUT QUERY *)
      MSG := 'INPUT QUERY';
      EDITOR(QUERY, QRYCNT, MSG);
    END;
    'S', 'S' : BEGIN
      TRANS; (* TRANSLATE QUERY *)
      IF PQFLAG <> NO THEN ANSWER := 'E';
    END;
    'P', 'P' : PRTORY;
    'D', 'D' : DSPLYNAM;
    'O', 'O' : FRTHAM;
    'E', 'e' : PQFLAG := NO;
  END; (* CASE *)
UNTIL ANSWER IN ['E', 'e'];
END; (* INPUTQUERY *)
BEGIN (* INITIALIZATION *)
END.
UNIT SOQUERY;
(* INPUTS A DRY AND SEARCHES A FILE *)

INTERFACE

USES APPLESTUFF, GLOBAL, SUPPORT, EDIT, TRANSLATE, EVALUATE, DRYINPUT;

PROCEDURE QUERYMAIN;

IMPLEMENTATION

VAR
CFLAG : INTEGER; (* YES-ALREADY ADV 1 FLD *)
DFLAG : INTEGER; (* YES-DONE *)
EDITED : INTEGER; (* YES-REC EDITED *)
EXTFLAG : INTEGER; (* YES-END QUERY MODE *)
FIRSTFLAG : INTEGER; (* YES-FILE JUST OPENED *)
FOPEN : INTEGER; (* YES-DFILE NOT OPEN *)
MARKFLAG : INTEGER; (* YES-REC MARKED/RECALLED *)
PRRTALL : INTEGER; (* YES-PRINT ALL RECS *)
PRRTHEADER : INTEGER; (* YES-PRINT HEADER *)
RCRDCNT : INTEGER; (* NUMBER RECORDS READ *)
SFLAG : INTEGER; (* YES-RECORDS RETRIEVED *)

PROCEDURE LISTREC;
(* DISPLAYS COPY OF RETRIEVED REC *)

VAR
ANSWER : CHAR; (* USER INPUT *)
COUNT : INTEGER; (* LOOP COUNTER *)
OKSET : SETOFCHAR; (* OK INPUT *)
TEMPCHAR : CHAR; (* CHAR OF FLD *)

BEGIN
PAGE(OUTPUT); LINECOUNT := 0;
RESET(TFILE, 'PI-STAR: TEMP');
(* LIST # FLD *)
WRITE(OUTPUT, 'RECORD MARK: '); WRITE(TFILE, TEMPCHAR);
WRITE(OUTPUT, TEMPCHAR);
FOR COUNT := 1 TO FIELDCNT DO
BEGIN (* LIST EACH FLD *)
WRITE(OUTPUT, FIELDNAME(COUNT), ': '); PAGE;
END;
WHILE NOT(EOLN(TFILE)) DO
BEGIN
READ(TFILE, TEMPCHAR);
IF TEMPCHAR = DELIMITER THEN BEGIN
WRITE(OUTPUT); PAGE;
END;
END;
ELSE WRITE(OUTPUT,TEMPCHAR);
END;
READLN(TFILE);
END;
CLOSE(TFILE,NORMAL);
WRITE(OUTPUT,'TYPE <SPACE> TO CONTINUE');
OKSET := (SPACE);
GETANSWER(OKSET,ANSWER);
END; (* LISTREC *)

PROCEDURE PRTREC;
(* GENERATES HARDCOPY OF RETRIEVED REC *)
VAR
  ANSWER :CHAR; (* USER INPUT *)
  COUNT :INTEGER; (* LOOP COUNTER *)
  OKSET :SETOFCHAR; (* LEGAL INPUT *)
  TEMPCHAR :CHAR; (* CHAR OF FLD *)
BEGIN
  IF PRTREC = YES THEN BEGIN
    IF PRTHEADER = YES THEN BEGIN (* PRT HEADER *)
      WRITELN(PRINTER,'THE FOLLOWING RECORDS ARE ',RECNAMPLUR);
      WRITELN(PRINTER,'FROM THE FILE ',FILENAME);
      PRTHEADER := NO;
    END;
    WRITELN(PRINTER);
    RESET(TFILE,'PI-STAR: TEMP');
    FOR COUNT := 0 TO FIELDCNT DO BEGIN (* PRINT EACH FLD *)
      IF COUNT = 0 THEN WRITE(PRINTER,'RECORD MARK: '); 
      ELSE WRITELN(PRINTER,FIELDNAME,COUNT,' '); 
      WHILE NOT(EOLN(TFILE)) DO BEGIN
        READ(TFILE,TEMPCHAR);
        IF TEMPCHAR = DELIMITER THEN WRITELN(PRINTER) 
        ELSE WRITE(PRINTER,TEMPCHAR);
      END;
      IF COUNT = 0 THEN WRITELN(PRINTER);
    END;
    CLOSE(TFILE,NORMAL);
  END ELSE BEGIN
    WRITELN(OUTPUT,'PRINTER NOT ON-LINE'); PAGING;
    WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE'); PAGING;
  END;
END;
OKSET := [SPACE];
GETANSWER(OKSET, ANSWER);
END;
END; (* PRREC *)

PROCEDURE BRKFLDS;
(* TRANSFERS REC INTO SEPARATE FILES *)
VAR
  COUNT :INTEGER; (* LOOP COUNTER *)
  EXISTS :INTEGER; (* YES-FLD.PND EXISTS *)
  IORES :INTEGER; (* IORESULT *)
  TEMPCHAR :CHAR; (* TEMP CHAR *)
  TSTR :STRING[11]; (* TEMP CHAR *)
  NAMFILE :STRING[13]; (* FILE NAME *)
BEGIN
  RESET(RFILE,'PI-STAR:TEMP');
  FOR COUNT := 0 TO FIELDCNT DO
  BEGIN
    IF COUNT = 0
    THEN BEGIN
      (**I-**)
      RESET(TFILE,'PI-STAR:FLD,PND');
      IORES := IORESULT;
      IF IORES <> 0
      THEN BEGIN
        REWRITE(TFILE,'PI-STAR:FLD,PND');
        EXISTS := NO;
      END;
      ELSE EXISTS := YES;
      (**I+**)
    END;
    ELSE BEGIN
      TSTR := SPACE;
      TSTR[1] := FIELDID[COUNT];
      NAMFILE := CONCAT('PI-STAR:FLD.',TSTR);
      REWRITE(TFILE,NAMFILE);
    END;
    IF EXISTS = NO
    THEN BEGIN
      WHILE NOT(EOLN(RFILE)) DO
      BEGIN
        READ(RFILE,TEMPCHAR);
        WRITE(TFILE,TEMPCHAR);
      END;
      READLN(RFILE);
    END;
    ELSE BEGIN
      READLN(RFILE);
    END;
  END;
  CLOSE(TFILE,LOCK);
END;
CLOSE(RFILE,NORMAL);
END; (* BRKFLDS *)

PROCEDURE EDTREC;
(* ALLOWS ANY FLD OF CURR REC TO BE EDITED *)
VAR
   ANSWER :CHAR; (* USER INPUT *)
   COUNT :INTEGER; (* LOOP COUNTER *)
   DONE :INTEGER; (* YES- FINISHED EDIT *)
   EDITID :CHAR; (* FLD TO BE CORRECTED *)
   EDITIDS :STRING[62]; (* Flds to be corrected *)
   FIELDNO :INTEGER; (* FLD to be edited *)
   MSG :STRING[50]; (* Passed to editor *)
   NAMFILE :STRING[13]; (* File name *)
   NUMBCHAR :INTEGER; (* Numb chars in Fld *)
   OKSET :SETOFCHAR; (* OK input *)
   TEMPCHR :CHAR; (* Temp char *)
   TSTR :STRING[1]; (* Temp str *)
BEGIN
   PAGE(OUTPUT); LINECOUNT := 0;
   BRKFLDS;
   DONE := NO;
   REPEAT (* UNTIL DONE=YES *)
      WRITELN(OUTPUT); PAGING;
      WRITELN(OUTPUT,'ENTER THE IDS OF FIELDS TO BE EDITED'); PAGING;
      WRITELN(OUTPUT,'OR <RETURN> TO EXIT:'); PAGING;
      READLN(INPUTEDITIDS); PAGING;
      IF EDITIDS NULL THEN BEGIN (* EDIT FIELDS *)
         EDITED := YES;
         EDITID := EDITIDS[1];
         IF EDITID IN ['A'..'Z','0'..'9'] THEN BEGIN
            IF EDITID IN ['A'..'Z']
               THEN FIELDNO := ORD(EDITID) - ORD('A') + 1
               ELSE FIELDNO := ORD(EDITID) - ORD('0') + 1;
            IF FIELDNO > FIELDCNT
               THEN BEGIN (* ID OUT OF RANGE *)
                  IF FIELDCNT = 0
                     THEN WRITELN(OUTPUT,'ERROR- NO FIELDS EXIST');
                  ELSE WRITELN(OUTPUT,'NO FIELDS EXIST AFTER FIELD ',
                     FIELDNO); PAGING;
               END;
            PAGING;
            WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE'); PAGING;
            OKSET := SPACE;
            IF NOT EDITIDS[1] THEN GETANSWER(OKSET,ANSWER);
            EDITIDS := NULL;
         END;
         ELSE BEGIN
            TSTR := SPACE;
         END;
      END;
      WRITELN(OUTPUT); PAGING;
      WRITELN(OUTPUT,'ENTER THE IDS OF FIELDS TO BE EDITED'); PAGING;
      READLN(INPUTEDITIDS); PAGING;
      IF EDITIDS NULL THEN BEGIN (* EDIT FIELDS *)
         EDITED := YES;
         EDITID := EDITIDS[1];
         IF EDITID IN ['A'..'Z','0'..'9'] THEN BEGIN
            IF EDITID IN ['A'..'Z']
               THEN FIELDNO := ORD(EDITID) - ORD('A') + 1
               ELSE FIELDNO := ORD(EDITID) - ORD('0') + 1;
            IF FIELDNO > FIELDCNT
               THEN BEGIN (* ID OUT OF RANGE *)
                  IF FIELDCNT = 0
                     THEN WRITELN(OUTPUT,'ERROR- NO FIELDS EXIST');
                  ELSE WRITELN(OUTPUT,'NO FIELDS EXIST AFTER FIELD ',
                     FIELDNO); PAGING;
               END;
            PAGING;
            WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE'); PAGING;
            OKSET := SPACE;
            IF NOT EDITIDS[1] THEN GETANSWER(OKSET,ANSWER);
            EDITIDS := NULL;
         END;
      END;
      WRITELN(OUTPUT); PAGING;
      WRITELN(OUTPUT,'ENTER THE IDS OF FIELDS TO BE EDITED'); PAGING;
      READLN(INPUTEDITIDS); PAGING;
   END;
END;
NAMFILE := CONCAT('PI-STAR:FLD.', TSTR);
RESET(TFILE, NAMFILE);
(* GET FIELD[FIELDNO] FROM TFILE FOR EDITING *)
NUMBCHAR := 0;
WHILE NOT(EOF(TFILE)) DO
   BEGIN
      NUMBCHAR := NUMBCHAR + 1;
      READ(TFILE, TMPCHR);
      FIELD[NUMBCHAR] := TMPCHR;
   END;
CLOSE(TFILE, NORMAL);
(* EDIT FLD *)
MSG := CONCATC(EDIT ', FIELDNAME[FIELDNO], ' OF ', RECNAMSING);
EDITOR(FIELD, NUMBCHAR, MSG);
(* SAVE FLD TO TFILE *)
REWRITE(TFILE, NAMFILE);
END; (* EDIT FLD *)
DELETE(EDITIDS, 1, 1);
UNTIL LENGTH(EDITIDS) = 0;
END; (* EDIT FLD *)
END; (* EDITFLDS *)

BEGIN
PROCEDURE SAVEREC;
(* SAVES REC IF MODIFIED *)
VAR
   ANSWER : CHAR; (* USER INPUT *)
   CNT : INTEGER; (* LOOP COUNTER *)
   CORRDSK : INTEGER; (* YES- HAVE OK DISK *)
   COUNT : INTEGER; (* LOOP COUNTER *)
   DONE : INTEGER; (* YES- DONE *)
   IORES : INTEGER; (* IORESULT *)
   OKSET : SETOFCHAR; (* OK INPUT *)
   TMPCHR : CHAR; (* 1ST FLD OF HEADER *)
   TSTR : STRING[1]; (* TEMPORARY STRING *)
   NAMFILE : STRING[23]; (* FILE NAME *)
BEGIN
   IF EDITED = YES THEN BEGIN
      EDITED := NO;
      WRITELN(CFILE, 'P'); (* PERM MARK OLD REC *)
      (* WRITE 'C' (CORRECT) TO HEADER *)
      NAMFILE := CONCAT('PI-FILE:', FNAME, '.HEAD');
      RESET(TFILE, NAMFILE);
      WRITE(TFILE, 'C');
      CLOSE(TFILE, NORMAL);
      CORRDSK := NO;
   END; (* EDIT REC *)
END; (* SAVEREC *)
CLOSE (CFILE, NORMAL);  
(* WRITE REC TO FILE *)
WRITEFILE (FOPEN); 
CLOSE (DFILE, NORMAL);  
FOPEN := 'YES';  
(* OPEN AND REPOSITION CFILE *)
NAMFILE := CONCAT('PI-FILE:', FNAME, '.DATA');
RESET (CFILE, NAMFILE); 
FOR CNT := 1 TO RCRDCNT DO 
  FOR COUNT := 0 TO FIELDCNT DO 
    READLN (CFILE);  
FIRSTFLAG := 'YES';  
(* ERASE TEMP FLD FILES *)
FOR COUNT := 0 TO FIELDCNT DO 
  BEGIN 
    IF COUNT = 0 THEN reset (TFILE, 'PI-STAR:FLD.PND') 
    ELSE BEGIN 
      TSTR := SPACE;
      TSTR[1] := FIELDIDC[COUNT]; 
      NAMFILE := CONCAT('PI-STAR:FLD.', TSTR); 
      reset (TFILE, NAMFILE); 
    END;  
  END; 
CLOSE (TFILE, PURGE);  
END; 
reset (TFILE, 'PI-STAR:TEMP'); 
CLOSE (TFILE, PURGE);  
(* DON'T CONT UNTIL CORRECT DISK IS IN DRIVE *)
DONE := 'NO';  
REPEAT 
  CHKFILE (IORES); 
  IF IORES = 0 THEN BEGIN 
    NAMFILE := CONCAT('PI-FILE:', FNAME, '.HEAD'); 
    reset (TFILE, NAMFILE); 
    read (TFILE, TMPCHAR);  
    IF TMPCHAR = 'C' THEN BEGIN 
      CORRDSK := 'YES';  
      close (TFILE, NORMAL); 
      reset (TFILE, NAMFILE); 
      write (TFILE, SPACE); 
      end 
    ELSE BEGIN 
      writeln (OUTPUT); 
      writeln (OUTPUT, 'PUT ORIGINAL DISK IN DRIVE'); 
      writeln (OUTPUT, 'AND TYPE <SPACE> TO CONTINUE'); 
      OKSET := SPACE;
      getanswer (OKSET, ANSWER); 
      close (TFILE, NORMAL); 
      end;  
  END; 
END;
UNTIL CORRDSK = YES;
END
ELSE IF MARKFLAG = YES
THEN BEGIN
  RESET(TFILE,'PI-STAR:FLD.PND');
  READ(TFILE, TMPCHAR);
  WRITELN(CFILE, TMPCHAR);
  CFLAG := YES;
  CLOSE(TFILE, PURGE);
  MARKFLAG := NO;
END;
END; (* SAVEREC *)

PROCEDURE MOREDISKS;
(* CALLED WHEN EOF IS TRUE ON THE SRCH DISK *)
VAR
  ANSWER :CHAR; (* USER INPUT *)
  IORES :INTEGER; (* IORESULT *)
  OKSET :SETOFCHAR; (* OK INPUT *)
  NAMFILE :STRING[23];(* FILE NAME *)
BEGIN
  WRITELN(OUTPUT); PAGING;
  WRITELN(OUTPUT,'END OF DISK REACHED.'); PAGING;
  WRITELN(OUTPUT,'ARE THERE ANY MORE DISKS TO SEARCH?'); PAGING;
  WRITE(OUTPUT,.'->' );
  OKSET := ['Y','N','y','n'];
  GETANSWER(OKSET,ANSWER);
 WRITELN(OUTPUT,ANSWER); PAGING;
  CLOSE(NFILE,NORMAL);
  CLOSE(CFILE,NORMAL);
  IF ANSWER IN ['Y','y']
  THEN BEGIN
    WRITELN(OUTPUT); PAGING;
    WRITELN(OUTPUT,'INSERT NEXT DISK AND'); PAGING;
    WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE'); PAGING;
    WRITELN(OUTPUT,'OR <E> TO EXIT'); PAGING;
    OKSET := [SPACE,'E','e'];
    GETANSWER(OKSET,ANSWER);
    IF ANSWER = SPACE
    THEN BEGIN
      CHKFILE(IORES);
      IF IORES <> 0
      THEN DFLAG := YES
      ELSE BEGIN (* OPEN FILE *)
        NAMFILE := CONCAT('PI-FILE:','FNAME','.DATA');
        RESET(CFILE,NAMFILE);
        RESET(NFILE,NAMFILE);
      END;
    END
    ELSE DFLAG := YES;
END
ELSE DFLAG := YES;
PROCEDURE MOVEC;
(* ADV CFiLE TO NEXT REC *)
VAR
  CNT  :INTEGER; (* FIELD PTR *)
  COUNT :INTEGER; (* LOOP CNTR *)
BEGIN
  IF FIRSTFLAG <> YES
  THEN BEGIN
    IF CFLAG = YES
    THEN BEGIN
      CFLAG := NO;
      CNT := 1;
    END
    ELSE CNT := 0;
    FOR COUNT := CNT TO FIELDCNT DO
      READLN(CFILE);
  END
  ELSE FIRSTFLAG := NO;
END; (* MOVEC *)

PROCEDURE SEARCH;
(* READS REC FROM CURR FILE + PREPARES *)
(* TO COMPARE REC WITH QUERY. *)
VAR
  CNT  :INTEGER; (* LOOP COUNTER *)
  COUNT :INTEGER; (* LOOP COUNTER *)
  DOTCNT :INTEGER; (* DOT COUNT/LINE *)
  NUMCHAR :INTEGER; (* NUM CHARAS IN FLD*)
  RESULT :INTEGER; (* RET BY EXPREVAL *)
  TEMPCHAR :CHAR; (* CHAR OF FLD *)
  TSTR  :STRING[11]; (* TEMP STR *)
  NAMFILE :STRING[13]; (* FILE NAME *)
BEGIN
  SFLAG := NO;
  DFLAG := NO;
  DOTCNT := 0;
  PAGE(OUTPUT); LINECOUNT := 0;
  WRITELN(OUTPUT, 'SEARCHING');
  REPEAT (* UNTIL REC IS TO BE RET OR NO RECORDS *)
    IF EOF(NFILE)
    THEN MOREDISKS;
    IF DOTCNT < 40
    THEN DOTCNT := DOTCNT + 1
    ELSE BEGIN
      WRITELN(OUTPUT);
      DOTCNT := 1;
    END;
  IF DFLAG <> YES
  THEN BEGIN
    (* PROCESSING CODE *)
  END;
WRITE(OUTPUT,'."');
(* IF REC IS NOT PERM MARKED- PUT IN TFILE *)
READLN(NFILE,TEMPCHAR);
IF TEMPCHAR <> 'P'
THEN BEGIN
  REWRITE(TFILE,'PI-START:TEMP');
  WRITELN(TFILE,TEMPCHAR);
  FOR COUNT := 1 TO FIELDCNT DO
BEGIN
    WHILE NOT(EOLN(NFILE)) DO BEGIN
      READ(NFILE,TEMPCHAR);
      WRITE(TFILE,TEMPCHAR);
    END;
    WRITELN(TFILE);
    READLN(NFILE);
  END;
  CLOSE(TFILE,LOCK);
(* COMPARE REC TO QRY *)
EXPREVAL (RESULT,EXTFLAG);
IF RESULT = TREW
THEN IF PRTALL = YES
    THEN BEGIN
      PRTREC;
      MOVEC;
    END
ELSE SFLAG := YES
ELSE IF EXTFLAG = YES
    THEN DFLAG := YES
ELSE MOVEC;
END
ELSE FOR COUNT := 1 TO FIELDCNT DO
READLN(NFILE);
RCRDCNT := RCRDCNT + 1;
END;
UNTIL (SFLAG = YES) OR (DFLAG = YES);
END;'(* SEARCH *)

PROCEDURE QUERYSEARCH:
(* DISPLAYS MENU: SRCH FILE, MANIPULATE RETRIEVED *)
(* RECS. SAVE EDITED RECS. *)
VAR
  ANSWER :CHAR; (* USER RESPONSE *)
  DONE :INTEGER; (* YES- DONE W/ QUERY *)
  FRSTSRCH :INTEGER; (* YES- FIRST TIME SRCH *)
  NODISK :INTEGER; (* YES- NO PI-FILE DISK *)
  OKSET :SETOFCHAR; (* OK INPUT *)
  NAMFILE :STRING[255]; (* FILE NAME *)

PROCEDURE INITIAL;
BEGIN
BEGIN ( * QUERYSEARCH *)
INITIAL;
PAGE(OUTPUT); LINECOUNT := 0;
(* OPEN FILE *)
NAMFILE := CONCAT('PI-FILE: ',FNAME,'.DATA');
RESET(CFILE,NAMFILE);
RESET(NFILE,NAMFILE);
(* SET PRTALL *)
GOTOXY(0,5);
WRITELN (OUTPUT, 'READY TO BEGIN SEARCH PROCESSING');
WRITELN(OUTPUT)="#PAGING;";
WRITELN(OUTPUT, 'PRINT ALL RETRIEVED RECORDS?');
IF ANSWER IN ['Y','y'] THEN 
  PRTALL := YES 
ELSE 
  PRTALL := NO;
REPEAT (* UNTIL DONE = YES *)
(* DISPLAY MENU *)
PAGE(OUTPUT); LINECOUNT := 0;
GOTOXY(0,5);
WRITELN(OUTPUT, 'SEARCH FILE');
WRITELN(OUTPUT, 'SEARCH FILE');
GOTOXY(0,10);
WRITELN(OUTPUT, 'SELECT:');
WRITELN(OUTPUT, 'BEGIN SEARCH');
WRITELN(OUTPUT, 'LIST RECORD');
WRITELN(OUTPUT, 'PRINT RECORD');
WRITELN(OUTPUT, 'EDIT RECORD');
WRITELN(OUTPUT, 'RECALL MARKED RECORD');
WRITELN(OUTPUT, 'ABORT CURRENT SEARCH');
WRITELN(OUTPUT, 'TERMINATE SEARCH MODE');
IF SFLAG = YES THEN BEGIN 
  OKSET := ['B','L','P','E','M','R','A','T',
            'b','l','p','e','m','r','a','t'];
  WRITELN(OUTPUT, 'RECORD FOUND');
  WRITE(OUTPUT,'->');
END
ELSE BEGIN
  IF FRSTSRCH = YES
  THEN BEGIN
    FRSTSRCH := NO;
    OKSET := ['B','A','T','b','a','t'];
  END;
  ELSE OKSET := ['A','T','a','t'];
END;
GETANSWER(OKSET, ANSWER);
WRITELN(OUTPUT, ANSWER);
CASE ANSWER OF
  'B','b' : BEGIN
    SAVEREC; (* SAVE MOD REC *)
    MOVEC; (* ADV CFILE *)
    SEARCH; (* CONT SRCH *)
    END;
  'L','l' : LISTREC;
  'P','p' : PRTREC;
  'E','e' : EDTREC;
  'M','m' : BEGIN (* MARK REC *)
    REWRITE(TFILE,'PI-STAR:FLD.PND');
    WRITE(TFILE,'M');
    MARKFLAG := YES;
    CLOSE(TFILE,LOCK);
    END;
  'R','r' : BEGIN (* RECALL REC *)
    REWRITE(TFILE,'PI-STAR:FLD.PND');
    WRITE(TFILE,'N');
    MARKFLAG := YES;
    CLOSE(TFILE,LOCK);
    END;
  'A','a' : BEGIN
    SAVEREC; (* SAVE MOD REC *)
    DONE := YES;
    END;
  'T','t' : BEGIN
    SAVEREC; (* SAVE MOD REC *)
    DONE := YES;
    EXTFLAG := YES;
    END;
END; (* CASE *)
UNTIL DONE = YES;
(* CLOSE FILE *)
CLOSE(NFILE,NORMAL);
CLOSE(CFILE,NORMAL);
END; (* QUERYSEARCH *)

PROCEDURE QUERYMAIN;
(* THE EXTERNAL PROC FOR SQUERY. IT FIRST CALLS *)
(* INPUTQUERY. IF POFLAG IS YES, QUERYSEARCH IS CALLED *)
(* THIS REPEATS UNTIL POFLAG= NO OR EXTFLAG= YES. *)
VAR
  ANSWER : CHAR; (* USER INPUT *)
  IORES : INTEGER; (* IORESULT *)
  OKSET : SETOFCHAR; (* OK INPUT *)
BEGIN
  PAGE(OUTPUT); LINECOUNT := 0;
  GOTOXY(0,3);
  WRITELN(OUTPUT,'- SEARCH -':27);
  GOTOXY(0,20);
  WRITELN(OUTPUT,'TYPE <SPACE> TO CONTINUE':29);
  WRITELN(OUTPUT,'<E> TO EXIT SEARCH':28);
  OKSET := ['E', 'e', SPACE];
  GETANSWER(OKSET,ANSWER);
  PAGE(OUTPUT); LINECOUNT := 0;
  IF ANSWER = SPACE THEN BEGIN
    (* GET HEADER *)
    GETHEADER;
    ORYCNT := 0;
    REPEAT (* UNTIL PDFLAG = NO OR EXTFLAG = YES *)
      (* INPUT AND PROCESS ORY *)
      INPUTQUERY;
      (* SRCH IF THERE IS A VALID POSTFIX ORY *)
      IF PDFLAG = YES THEN QUERYSEARCH;
      UNTIL (PDFLAG = NO) OR (EXTFLAG = YES);
    END;
  END; (* QUERYMAIN *)
BEGIN
  (* INIT *)
END.
UNIT EVALUATE;

(* CONTAINS THE EXPRESSION EVALUATION STUFF *)

INTERFACE

USES GLOBAL,APPLESTUFF,SUPPORT;

PROCEDURE EXPREVAL(VAR RESULT,EXTFLAG: INTEGER);

IMPLEMENTATION

VAR

MSG : STRING[76]; (* ERROR MESSAGE *)
NUMBCHAR : INTEGER; (* NUM OF CHAR$ IN FIELD *)

PROCEDURE EXPREVAL; (* VAR RESULT,EXTFLAG: INTEGER *)

(* EXPRESSION EVALUATOR TAKES QUERY - THE POSTFIX FORM *)
(* OF THE QUERY AND REDUCES IT TO A 'TRUE' (THE CURRENT *)
(* RECORD MATCHES THE QUERY) OR A 'FALSE' (THE CURRENT *)
(* RECORD DOESN'T MATCH THE QUERY). *)

VAR

ANSWER : CHAR; (* USER INPUT *)
CURRFLD : INTEGER; (* THE CURRENT FIELD FROM TFILE *)
EMPTY : INTEGER; (* IF YES - THE STACK IS EMPTY *)
FLDTOKEN : TOKENS; (* THE TOKEN TELLS FIELD TO BE USED*)
OKSET : SETOFCHAR; (* OK INPUT *)
OPERTOKEN : TOKENS; (* THE OPERATOR BEING USED *)
PQRYCNT : INTEGER; (* TELLS THE POSITION IN PQURY *)
PTKN : TOKENS; (* TOKEN JUST POPPED *)
STACK : PACKED ARRAY[0..90] OF TOKENS; (* HOLDS OPERAND TOKENS, 'T', 'F' *)
STACKPTR : INTEGER; (* INDICATES EMPTY PLACE IN STACK *)
STRTOKEN : TOKENS; (* THE TOKEN TELLS WHICH SEARCH STR *)
TKN : TOKENS; (* THE CURRENT TOKEN *)
TOKEN1 : TOKENS; (* TEMP TOKEN *)
TOKEN2 : TOKENS; (* TEMP TOKEN *)

PROCEDURE RELEXPREVAL:

(* RELATIONAL EXPRESSION EVALUATOR GETS THE FIELD INDICATED *)
(* BY 'FLDTOKEN' AND CALLS ONE OF TWO EVALUATION ROUTINES - *)
(* 'EONE' IF OPERTOKEN IS = OR <>, OR, 'LELTGEGT' IF *)
(* OPERTOKEN IS <, <=, >, >=). *)

VAR

COUNT : INTEGER; (* LOOP COUNTER *)
FIELDNO : INTEGER; (* THE FIELD IDENTIFIED BY FLDTOKEN *)
TEMPCHAR : CHAR; (* TEMPORARY CHARACTER *)
TMPSTR : STRING[11]; (* TEMPORARY STRING *)
PROCEDURE LELTGEGT;
(* LELTGEGT COMPARES THE SEARCH STRING TO AN EQUAL NUMBER *)
(* OF CHARACTERS FROM THE BEGINNING OF THE FIELD. *)
VAR
COUNT : INTEGER; (* LOOP COUNTER *)
FLDSTR : STRING[255]; (* CHARACTERS FROM FIELD *)
LENSTR : INTEGER; (* LENGTH OF SrchStr *)
BEGIN (* LELTGEGT *)
FLDSTR := NULL;
COUNT := 0;
TMPSTR := SPACE;
LENSTR := LENGTH(SRCHSTRSTR_TOKEN);
WHILE (COUNT < LENSTR) AND (COUNT < NUMBCHAR) DO
BEGIN (* BUILD FIELD STRING *)
COUNT := COUNT +1;
TMPSTR := SPACE;
TMPSTR[1] := FIELD[COUNT];
FLDSTR := CONCAT(FLDSTR, TMPSTR);
END;
(* DETERMINE VALUE OF RESULT *)
IF (OPERTOKEN = LT) AND (FLDSTR < SRCHSTRSTR_TOKEN)
THEN RESULT := TRUE
ELSE IF (OPERTOKEN = LE) AND (FLDSTR <= SRCHSTRSTR_TOKEN)
THEN RESULT := TRUE
ELSE IF (OPERTOKEN = GT) AND (FLDSTR > SRCHSTRSTR_TOKEN)
THEN RESULT := TRUE
ELSE IF (OPERTOKEN = GE) AND (FLDSTR >= SRCHSTRSTR_TOKEN)
THEN RESULT := TRUE
ELSE RESULT := FALSE;
END; (* LELTGEGT *)

PROCEDURE EQNE;
(* TESTS THE FIELD TO SEE IF THE SRCHSTR OCCURS ANYWHERE IN *)
(* THE FIELD. RETURNS AN APPROPRIATE VALUE DEPENDING ON *)
(* WHETHER OPERTOKEN WAS "=" OR "<>". *)
VAR
DONE : INTEGER; (* IF YES - THEN SEARCH DONE *)
FLDCNT : INTEGER; (* HOLDS CURRENT POSITION IN FLDSTR *)
MATCH : INTEGER; (* IF YES - SRCHSTR FOUND IN FIELD *)
MATCHPT : INTEGER; (* CHAR AFTER * MATCHED IN FIELD *)
STARPT : INTEGER; (* SHOWS WHERE CURRENT MATCH BEGAN *)
STRCNT : INTEGER; (* HOLDS CURRENT POSITION IN SRCHSTR *)
TMPSTR : STRING[1]; (* TEMPORARY STRING *)

PROCEDURE MATCHALL;
(* MATCHALL IS CALLED WHEN A WILDCARD OPERATOR IS ENCOUNTERED *)
(* IN THE SEARCH STRING. IT MATCHES THE REMAINDER OF *)
(* CHARACTERS IN THE CURRENT FIELD OR UNTIL THE CHARACTER *)
BEGIN
  STRCNT := STRCNT + 1; (* POINT TO CHARACTER AFTER THE '***' *)
  IF STRCNT <= LENGTH(SRCHSTR[STRTOKEN])
    THEN BEGIN
      WHILE (FLDCNT <= NUMBCHAR) AND
        (FIELD[FLDCNT] <> SRCHSTR[STRTOKEN, STRCNT]) DO
        FLDCNT := FLDCNT + 1;
        MATCHPT := FLDCNT + 1;
      END
      ELSE MATCH := YES;
    END; (* MATCHALL *)
  END; (* SRCHFLD *)
BEGIN (* EQNE *)
  MATCH := NO;
  DONE := NO;
  STRCNT := 1;
  FLDCNT := 1;
  STARTFT := 1;
  IF LENGTH(SRCHSTR[STRTOKEN]) <> 0
THEN BEGIN
  WHILE (FLDCNT <= NUMBCHAR) AND (MATCH = NO) DO
  BEGIN
    (* CALL SRCHFLD TO COMPARE SRCHSTR WITH FIELD *)
    SRCHFLD;
    /* IF MATCH HASN'T ALREADY BEEN SET BY MATCHALL THEN */
    /* DETERMINE IF THERE WAS A MATCH OR NOT */
    IF MATCH = NO
      THEN IF STRCNT > LENGTH(SRCHSTRSTRTOKEN)
          THEN IF FLDCNT > NUMBCHAR
              THEN MATCH := YES
              ELSE IF NOT(FIELD(FLDCNT) IN ['A'..'Z','0'..'9'])
                  THEN MATCH := YES;
    /* IF NOT DONE THEN PREPARE COUNTERS TO TRY AGAIN */
    IF (MATCH = NO) AND (FLDCNT <= NUMBCHAR)
      THEN BEGIN
        STRCNT := 1;
        IF SRCHSTRSTRTOKEN,STRCNT <> 'X'
          THEN BEGIN
            WHILE (FIELD[STARTPT] <> SPACE) AND
                (STARTPT <= NUMBCHAR) DO
              STARTPT := STARTPT + 1;
          END
          ELSE STARTPT := MATCHPT;
        END;
        FLDCNT := STARTPT;
        DONE := NO;
      END;
    END; (* WHILE *)
  END; (* THEN BEGIN *)
  ELSE MATCH := YES;
  /* DETERMINE RESULT TO BE RETURNED */
  IF (MATCH = YES) AND (OPERTOKEN = EQ)
    THEN RESULT := TRUE
  ELSE IF (MATCH = NO) AND (OPERTOKEN = NE)
    THEN RESULT := TRUE
  ELSE RESULT := FALSE;
END; (* EUNE *)

BEGIN (* RELEXPREVAL *)
/* GET A FIELD IF NECESSARY. ISSUE ERROR IF FIELD DOESN'T EXIST */
FIELDNO := ORD(FLDTOKEN) - 15;
IF FIELDNO <= FIELDCNT (* IT'S A LEGAL FIELD *)
  THEN BEGIN
    IF FIELDNO <> CURRFLD
      THEN BEGIN (* GET NEW FIELD *)
        RESET(TFILE,'PI-STAR;TEMP');
        (*POSITION TO PROPER FIELD *)
        IF FIELDNO <> 0
          THEN FOR COUNT := 0 TO FIELDNO - 1 DO
READLN(TFILE);
CURRFLD := FIELDNO;
NUMBCHAR := 0;
WHILE NOT(EOLN(TFILE)) DO
  BEGIN
    READ(TFILE, TEMPCHAR);
    IF TEMPCHAR = DELIMITER
      THEN TEMPCHAR := SPACE;
    NUMBCHAR := NUMBCHAR +1;
    FIELDNUMBCHARJ := TEMPCHAR;
  END;
CLOSE(TFILE,NORMAL);
END;

ELSE BEGIN (*ERROR*)
  PQFLAG := NO;
  TMPSTR := SPACE;
  TMPSTR := FIELDID[FIELDNO];
  MSG := CONCAT('FIELD ',TMPSTR,' DOESN'T EXIST');
END;

(* CALL PROPER ROUTINE *)
IF OPERTOKEN IN [EQ,NE]
  THEN EQNE
ELSE LELTGEGT;
END; (* RELEXPREVAL *)

PROCEDURE PUSH;
  (* PUSHES A TKN ONTO THE STACK *)
BEGIN
  IF STACKPTR < 90
    THEN BEGIN
      STACK[STACKPTR] := TKN;
      STACKPTR := STACKPTR + 1;
      EMPTY := NO;
    END
  ELSE BEGIN
    PQFLAG := NO;
    MSG := 'QUERY TOO COMPLEX';
  END;
END; (* PUSH *)

PROCEDURE POP;
  (* REMOVES A TKN FROM THE STACK *)
BEGIN
  STACKPTR := STACKPTR - 1;
  PTKN := STACK[STACKPTR];
  IF STACKPTR = 1
    THEN EMPTY := YES;
END; (* POP *)
BEGIN (* EXPREVAL *)
STACKPTR := 1;
EMPTY := YES;
CURRFLD := -1;
(* GET FIRST TOKEN OF QUERY *)
PQRYCNT := 1;
TKN := PQUERY(PQRYCNT);
WHILE TKN <> DELM DO
BEGIN
  IF TKN >= TOR
    THEN BEGIN (* TOKEN IS AN OPERATOR *)
      IF TKN < TNOT
        THEN BEGIN (* PROCESS RELATIONAL OP *)
          IF TKN = TRU
            THEN RESULT := TREW
            ELSE TKN := FALS;
          ELSE TKN := TRU
          OPERTOKEN := TKN;
          RELEXPREVAL; (* EVALUATES RELATIONAL EXPRESSIONS *)
        END;
      ELSE BEGIN (* POP STRTOKEN AND FLDTOKEN *)
        STRTOKEN := PTKN;
        POP;
        FLDTOKEN := PTKN;
      END;
    END;
  ELSE IF TKN = TNOT
    THEN BEGIN (* POP ONE OPERAND *)
      POP;
      TOKEN1 := PTKN;
      IF TOKEN1 = TRU
        THEN RESULT := FALS
        ELSE RESULT := TREW
    END;
ELSE BEGIN (* NOT *)
  IF TKN = TRU
    THEN RESULT := FALS
    ELSE RESULT := TREW
END;
END
END
END
ELSE BEGIN (* TOKEN IS OPERAND *)
  PUSH; (* OPERAND *)
IF POFLAG = YES
THEN BEGIN
(* GET NEXT TOKEN *)
  PORYCNT := PORYCNT + 1;
  TKN := POQUERY(PORYCNT);
END
ELSE TKN := DELM;
END; (* WHILE *)
(* GET FINAL RESULT *)
IF POFLAG = YES
THEN BEGIN
  FCIP;
  IF PTKN = TRU
  THEN RESULT := TREW
  ELSE RESULT := FALSA;
END
ELSE BEGIN
  WRITELN(OUTPUT); PAGING;
  WRITELN(OUTPUT, 'ERROR<< ', MSG); PAGING;
  WRITELN(OUTPUT, 'TYPE <SPACE> TO CONTINUE'); PAGING;
  OKSET := [SPACE];
  GETANSWER(OKSET, ANSWER);
  RESULT := FALSA;
  EXTFLAG := YES;
END;
END; (* EXPREVAL *)

BEGIN
(* INITIALIZATION *)
END.
UNIT SUPPORT;
(* CONTAINS PROCEDURES USED BY ALL OF PI-STAR *)

INTERFACE

USES APPESTUFF,GLOBAL;

PROCEDURE DATAOUT (OUTDATA:PACKEDARRAY; NUMCHAR: INTEGER);
PROCEDURE DISKADD (VAR NODISK: INTEGER);
PROCEDURE GETANSWER (OKSET: SETOFCHAR; VAR ANSWER: CHAR);
PROCEDURE GETHEADER;
PROCEDURE GETFNAME;
PROCEDURE ONLINE (VAR NODISK: INTEGER);
PROCEDURE PAGING;
PROCEDURE CHKFILE (VAR IORES: INTEGER);
PROCEDURE WRITEFILE (VAR FOPEN: INTEGER);

IMPLEMENTATION

PROCEDURE CHKFILE (*VAR IORES:INTEGER *)
(* CHECKS TO MAKE SURE THERE IS A FILE TO SEARCH *)
VAR
WAITCOUNT : INTEGER; (* HOLD CNT *)
WHOLEFILENAME : STRING[23]; (* FILE NAME *)
BEGIN
REWRITE(TFILE,'PI-STAR:TEMP');
CLOSE(TFILE,PURGE);
WHOLEFILENAME := CONCAT('PI-FILE:',FNAME,'.HEAD');
(*$I-*)
RESET(DFILE,WHOLEFILENAME);
IORES := IORESULT;
(*$I+*)
IF IORES <> 0
THEN BEGIN
WRITELN(OUTPUT);
WRITELN(OUTPUT,'ERROR - NO FILE ',FILENAME,' FOUND');
WAITCOUNT := 40;
REPEAT
WAITCOUNT := WAITCOUNT - 1;
NOTE(0,15);
UNTIL WAITCOUNT = 0;
END
ELSE CLOSE(DFILE,NORMAL); (* CHKFILE *)
END;

PROCEDURE GETFNAME;
(* GETFNAME INPUTS THE FILENAME AND CREATES A 10 CHAR *)
(* FNAME FROM FILENAME. USED WHEN CREATING DISK FILES *)
VAR
COUNT : INTEGER; (* LOOP COUNTER *)
INFO : STRING[30]; (* TEMP FILENAME *)
TEMPCHAR : STRING[1]; (* TEMP CHARACTER *)
BEGIN
READLN(INPUT, INFO);
IF INFO <> NULL THEN BEGIN
FILENAME := INFO;
(* GET FIRST 10 ALPHANUMERIC CHARS OF FILENAME *)
COUNT := 0;
TEMPCHAR := SPACE;
FNAME := NULL;
REPEAT
COUNT := COUNT + 1;
TEMPCHAR[1] := FILENAME[COUNT];
IF TEMPCHAR[1] IN ['A'..'Z','0'..'9']
THEN FNAME := CONCAT(FNAME, TEMPCHAR);
UNTIL (COUNT = LENGTH(FILENAME)) OR (LENGTH(FNAME) = 10);
IF FNAME = NULL (* NO ALPHANUMERIC CHARS IN FILENAME *)
THEN FNAME := 'DUMMYNAME';
END; /* GETFNAME *)
END; /* GETHEADER *)
PROCEDURE GETHEADER;
(* GETS FILE HEADER *)
VAR
COUNT : INTEGER; (* LOOP COUNTER *)
WHOLEFILENAME : STRING[22]; (* FILE NAME *)
BEGIN
WHOLEFILENAME := CONCAT('PI-FILE:', FNAME, '.HEAD');
RESET(TFILE, WHOLEFILENAME);
READLN(TFILE); (* SKIP FIRST FIELD *)
READLN(TFILE, FILENAME);
READLN(TFILE, RECNAMING);
READLN(TFILE, RECNAMPLUR);
READLN(TFILE, FIELDCNT);
FOR COUNT := 1 TO FIELDCNT DO
READLN(TFILE, FIELDDATA[COUNT]);
CLOSE(TFILE, NORMAL);
END; /* GETHEADER *)
PROCEDURE PAGING;
(* PAGING STOPS SCREEN PRINTING WHEN IT IS FULL. PROCESSING *)
(* CONTINUES WHEN A HOLD TIMES OUT (ABOUT 5 SECONDS) OR WHEN *)
(* AN ESC CHAR IS TYPED. THE USER ENABLES THE TIME-OUT *)
(* FEATURE BY TYPING ANY CHARACTER. EXECUTION CAN ONLY BE *)
(* RESUMED WHEN THE USER TYPES ANY CHARACTER. *)
CONST
CONTSTR = 'CONTINUE CONTINUE';
WAITSTR = 'WAITING WAITING';
SPACESTR :='

VAR
   ANSWER : CHAR; (* USER RESPONSE *)
   WAITCOUNT : INTEGER; (* COUNT FOR HOLD *)
BEGIN
   LINECOUNT := LINECOUNT + 1;
   IF LINECOUNT = 23 (* IF 23 LINES PRINTED, TIME TO WAIT *)
     THEN BEGIN
       LINECOUNT := 0;
       GOTOXY(0,23);
       WRITE(OUTPUT,CONTSTR:80);
       WRITE(OUTPUT,BELL);
       WAITCOUNT := 40;
       REPEAT (*UNTIL TIME EXPIRES OR USER ENTERS A KEY *)
         WAITCOUNT := WAITCOUNT - 1;
         NOTE(0,15);
       UNTIL (WAITCOUNT = 0) OR (KEYPRESS);
       IF KEYPRESS
         THEN BEGIN
           READ(0,ANSWER);
           IF ANSWER <> ESC (* HOLD UNTIL USER INPUT *)
             THEN BEGIN
               GOTOXY(0,23);
               WRITE(OUTPUT,WAITSTR:80);
               READ(0,ANSWER);
             END;
         END;
     END;
   GOTOXY(0,23);
   WRITE(OUTPUT,SPACESTR:80);
   GOTOXY(0,23);
END; (* PAGEING *)

PROCEDURE DATAOUT; (* OUTDATA:PACKEDARRAY; NUMBCHAR:INTEGER *)
   (* OUTPUTS DATA A CHARACTER AT A TIME. OUTDATA CONTAINS *)
   (* THE DATA TO BE OUTPUT FROM 1 TO NUMBCHAR. *)
VAR
   COUNT : INTEGER; (* LOOP COUNTER *)
BEGIN
   FOR COUNT := 1 TO NUMBCHAR DO
     BEGIN
       IF OUTDATACOUNT = DELIMITER
         THEN BEGIN
           WRITELN(OUTPUT); PAGING;
         END
       ELSE WRITE(OUTPUT,OUTDATACOUNT);
     END;
END; (* DATAOUT *)

PROCEDURE GETANSWER; (* OKSET: SETOFCHAR; VAR ANSWER: CHAR *)
   (* THIS PROCEDURE ONLY ACCEPTS USER INPUT OF SPECIFIED *)
(* SET. ANY OTHER RESPONSE CAUSES A 'BEEP' UNTIL A *)
(* CHAR FROM THE OKSET IS ENTERED. *)

BEGIN
  REPEAT
    READ (KEYBOARD, ANSWER);
    IF NOT (ANSWER IN OKSET) THEN WRITE (OUTPUT, BELL);
    UNTIL (ANSWER IN OKSET);
END;
(* GETANSWER *)

PROCEDURE ONLINE; (* VAR NODISK: INTEGER *)
(* CHECKS TO MAKE SURE PI-FILE IS ONLINE *)
VAR
  ANSWER: CHAR; (* USER SINGLE CHAR INPUT *)
  IORES: INTEGER; (* I/O RESULT *)
  OKSET: SET OF CHAR; (* LEGAL INPUT CHARS *)
BEGIN
  NODISK := NO;
  REPEAT
    (* CHECK FOR PI-FILE ON-LINE *)
    (**-** (* DISABLE I/O ERRORS *)
      RESET (DFILE, 'PI-FILE:');
      IORES := IORES;
    (**+** (* ENABLE I ERRORS *)
      IF IORES <> 0 THEN BEGIN
        WRITELN (OUTPUT, 'PI-FILE VOLUME NOT ONLINE.');
        WRITELN (OUTPUT, 'INSERT CORRECT VOLUME AND TYPE <SPACE>');
        WRITELN (OUTPUT, 'OR TYPE <E> TO EXIT TO MAIN MENU');
        OKSET := ['E', 'e', SPACE];
        GETANSWER (OKSET, ANSWER);
        PAGE (OUTPUT); LINECOUNT := 0;
        IF ANSWER IN ['E', 'e'] THEN NODISK := YES;
      END ELSE CLOSE (DFILE, NORMAL);
    END
    UNTIL (ANSWER IN ['E', 'e']) OR (IORES = 0);
END; (* ONLINE *)

PROCEDURE DISKADD; (* VAR NODISK: INTEGER *)
(* SETS UP HEADER AND DATA FILE FOR EMPTY PI-FILE DISK *)
VAR
  COUNT: INTEGER; (* LOOP COUNTER *)
  IORES: INTEGER; (* IORES *)
  SPACE2: STRING[2]; (* PLACE HOLDER *)
  WAITCOUNT: INTEGER; (* HOLD *)
  WHOLEFILENAME: STRING[233]; (* COMPLETE FILE NAME *)
BEGIN
  (* CHECK FOR FILE OF SAME FNAME ALREADY EXISTING *)
  REWRITE (TFILE, 'PI-STAR:TEMP');
  CLOSE (TFILE, PURGE);
WHOLEFILENAME := CONCAT('PI-FILE:', FNAME, '.HEAD');
(*$ I-*)
RESET(DFILE, WHOLEFILENAME);
IRES := IORES;
(*$I*+)
IF IORES = 0 THEN BEGIN
  WRITELN(OUTPUT, 'FILE ', FNAME, ' ALREADY EXISTS');
  WAITCOUNT := 0;
  REPEAT
    WAITCOUNT := WAITCOUNT - 1;
    NOTE(0, 15);
  UNTIL WAITCOUNT = 0;
  CLOSE(DFILE, NORMAL);
  NODISK := YES;
  END
ELSE BEGIN
  (* CHECK PI-FILE ONLINE *)
  ONLINE(NODISK);
  IF NODISK = NO THEN BEGIN
    SPACE2 := ' 
    WHOLEFILENAME := CONCAT('PI-FILE:', FNAME, '.HEAD');
    REWRITE(TFILE, WHOLEFILENAME);
    WRITELN(TFILE, SPACE2);
    WRITELN(TFILE, RECNAMING);
    WRITELN(TFILE, RECNAMPLUR);
    WRITELN(TFILE, FIELDcnt);
    FOR COUNT := 1 TO FIELDcnt DO
      WRITELN(TFILE, FIELDNAMCOUNT);
    CLOSE(TFILE, LOCK);
    (* CREATE INFORMATION FILE *)
    WHOLEFILENAME := CONCAT('PI-FILE:', FNAME, '.DATA');
    REWRITE (DFILE, WHOLEFILENAME);
    CLOSE(DFILE, LOCK);
  END;
END;
END; (* DISKADD *)

PROCEDURE NEXTDISK(VAR FORGETIT: INTEGER);
  (* PUTS A NEW DISK ON-LINE *)
VAR
  ANSWER : CHAR; (* USER SINGLE CHAR INPUT *)
  IORES : INTEGER; (* I/O RESULT *)
  NODISK : INTEGER; (* IF YES- NO PI-FILE DISK *)
  OKSET : SETOFCHAR; (* LEGAL INPUT CHARS *)
  WHOLEFILENAME : STRING[23]; (* COMPLETE FILENAME *)
BEGIN
  WRITELN(OUTPUT, 'INSERT NEXT DISK AND'); PAGING;
  WRITELN(OUTPUT, 'TYPE <SPACE> TO CONTINUE'); PAGING;
  OKSET := [SPACE];
GETANSWER(OKSET, ANSWER);
WRITELN(OUTPUT);
WRITELN(OUTPUT,'TYPE <N> FOR NEW DISK'); PAGING;
WRITELN(OUTPUT,'OR TYPE <E> FOR EXISTING DISK'); PAGING;
WRITE(OUTPUT,'->');
OKSET := ['N','n','E','e'];
GETANSWER(OKSET, ANSWER);
WRITELN(OUTPUT,ANSWER); PAGING;
WRITE(OUTPUT,'->'); OKSET := 'N', 'n', 'E', 'e';
GETANSWER(OKSET, ANSWER);
WRITELN(OUTPUT); PAGING;
IF ANSWER IN ['N','n']
THEN BEGIN
DISKADD(NODISK);
IF NODISK = NO
THEN BEGIN
WHOLEFILENAME := CONCAT('PI-FILE:',FNAME,'.DATA');
RESET(DFILE,WHOLEFILENAME);
FORGETIT := NO;
END
END
ELSE BEGIN
CHKFILE(IORES);
IF IORES = O
THEN BEGIN
FORGETIT := NO;
(* OPEN FILE *)
WHOLEFILENAME := CONCAT('PI-FILE:',FNAME,'.DATA');
RESET(DFILE,WHOLEFILENAME);
(* POSITION TO END OF DATA FILE *)
RDLNCNT := O;
WHILE NOT(EOF(DFILE)) DO BEGIN
READLN(DFILE);
RDLNCNT := RDLNCNT + 1;
END;
END;
END; (* NEXTDISK *)

PROCEDURE WRITEFILE; (* FOPEN: INTEGER *)
(* WRITES A NEW OR EDITED RECORD TO THE ACTIVE FILE *)
VAR
ANSWER : CHAR; (* USER SINGLE CHAR INPUT *)
COUNT : INTEGER; (* LOOP COUNTER *)
CNT : INTEGER; (* LOOP COUNTER *)
FORGETIT : INTEGER; (* IF YES- DO NOT ADD RECORD *)
IORES : INTEGER; (* I/O RESULT *)
NUMCHAR : INTEGER; (* NUMBER OF CHARS IN FIELD *)
OKSET : SETOFCHAR; (* LEGAL INPUT CHAR *)
RCOUNT : INTEGER; (* LOOP COUNTER *)
TEMPCHAR : CHAR; (* TEMPORARY CHAR *)
TFILENAME : STRING[1]; (* FILE NAME *)
TSTR : STRING[1]; (* TEMP STRING *)
WHOLEFILENAME : STRING[256]; (* COMPLETE FILENAME *)
BEGIN
  IF FOPEN = YES
  THEN BEGIN
    (* OPEN FILE *)
    WHOLEFILENAME := CONCAT('PI-FILE:', FNAME, '.DATA');
    RESET(DFILE, WHOLEFILENAME);
    FOPEN := NO;
    (* POSITION TO END OF DATA FILE *)
    RDLNCT := 0;
    WHILE NOT(EOF(DFILE)) DO
      BEGIN
        READLN (DFILE);
        RDLNCT := RDLNCT + 1;
      END;
      ++
    END;
    WRITELN(OUTPUT, 'PLEASE WAIT -');
    WRITELN(OUTPUT, 'WRITING RECORD TO FILE');
    WRITELN(OUTPUT);
    FORGETIT := NO;
    IORES := 0;
    CNT := 0;
    REPEAT (* WRITE RECORD TO FILE *)
      IF CNT = 0
      THEN BEGIN
        RESET(TFILE, 'PI-STAR:FLD.PND')
      ELSE BEGIN
        TSTR := SPACE;
        TFILENAME := CONCAT('PI-STAR:FLD.', TSTR);
        RESET(TFILE, TFILENAME);
      END;
      CNT := CNT + 1;
    END;
    NUMBCHAR := 0;
    WHILE NOT(EOF(TFILE)) DO
      BEGIN
        NUMBCHAR := NUMBCHAR + 1;
        READ(TFILE, TEMPCHAR);
        FIELD[NUMBCHAR] := TEMPCHAR;
      END;
    CLOSE(TFILE, NORMAL);
    THEN BEGIN
      COUNT := 0;
      REPEAT (* WRITE FIELD TO FILE *)
        COUNT := COUNT + 1;
        (**I-**) (* DISABLE I/O ERRORS *)
        WRITE(DFILE, FIELD[COUNT]);
        IORES := IORESULT;
        (**I++*) (* ENABLE I/O ERRORS *)
        IF IORES = 0
        THEN FORGETIT := YES;
      UNTIL (COUNT = NUMBCHAR) OR (FORGETIT = YES);
IF FORGETIT <> YES
THEN (* WRITE DELIMITER TO SEPARATE FIELDS *)
   (**I-**) WRITELN(DFILE);
   (**I+**)
IF FORGETIT = YES
THEN BEGIN (* DISK FULL *)
   (* RESTORE DATA FILE ORIGINAL *)
   RESET(DFILE);
   FOR RCOUNT := 1 TO RDLNCNT DO
      READLN(DFILE);
   CLOSE(DFILE, CRUNCH);
   WRITELN(OUTPUT,'DISK FULL'); PAGING;
   WRITELN(OUTPUT,'TYPE <A> FOR ANOTHER DISK OR'); PAGING;
   WRITELN(OUTPUT,'TYPE <E> TO EXIT WITHOUT ADDING RECORD'); PAGING;
   WRITE(OUTPUT,'->');
   OKSET := ['A','a','E','e'];
   GETANSWER(OKSET, ANSWER);
   WRITELN(OUTPUT, ANSWER); PAGING;
   IF ANSWER IN ['A','a']
   THEN BEGIN
      CNT := 0;
      NEXTDISK(FORGETIT);
   END;
END; (* DISK FULL *)
UNTIL (CNT = FIELDCNT + 1) OR (FORGETIT = YES);
IF FORGETIT = NO
THEN RDLNCNT := RDLNCNT + FIELDCNT + 1;
END; (* WRITEFILE *)
UNIT GLOBAL:
(* CONTAINS DECLARATIONS AND INITIALIZATIONS *)
(* USED BY THE ENTIRE PI-STAR SYSTEM. *)

INTERFACE

CONST
SPACE = ' ';
YES = 1;
NO = 0;
NULL = '0';
TREW = 1;
FALS = 0;

TYPE
SETOFCHAR = SET OF CHAR;
PACKEDARRAY = PACKED ARRAY[1..512] OF CHAR;
TOKENS = (STR1 ,STR2 ,STR3 ,STR4 ,STR5 ,STR6 ,STR7 , (* SEARCH STRINGS *)
STR8 ,STR9 ,STR10,STR11,STR12,STR13,STR14, STR15,
IDPND,IDA,IDB,IDC,IDD,IDE,IDF,IDG,IDI,IDL, # FIELD IDENTIFIERS *)
IDJ,IDL,IDD,IDN,IDO,IDP,IDQ,IDS,
IDT,IDD,IDE,IDI,IDI,IDL,IDL,IDL,IDL,
IDM,IDS,IDO,IDO,IDO,IDO,IDO,IDO,
DELIM, (* DELIMITER *)
LPAR, (* LEFT PARENTHESIS *)
TDR,TAND,TNOT, (* LOGICAL OPS *)
EO,NE,LT,GT,LE,GE, (* RELATIONAL OPS *)
RPAR, (* RIGHT PARENTHESIS *)
TRU,FALS); (* TRUE, FALSE *)

VAR
(* CONSTANTS *)
BACKSP : CHAR; (* BACKSPACE *)
BELL : CHAR; (* BELL *)
CTRLC : CHAR; (* CONTROL-C EOF CHAR *)
DELMITER : CHAR; (* CR SUBSTITUTE *)
ESC : CHAR; (* USED TO CONTINUE *)
RARROW : CHAR; (* FORWARD KEY (->) *)
(* GLOBALS *)
CFILE : INTERACTIVE; (* CURRENT RECORD FILE *)
DFILE : TEXT; (* ADD RECORD FILE *)
FIELD : PACKED ARRAY[1..512] OF CHAR; (* FIELD OF RECORD *)
FIELDCNT : INTEGER; (* NUMBER OF FIELDS/RECORD *)
FIELDID : PACKED ARRAY[1..36] OF CHAR; (* FIELD IDS A..Z,0..9 *)
FIELDNUM : PACKED ARRAY[1..36] OF STRING[30];
FILENAME : STRING[30]; (* FILE NAME *)
FNAME : STRING[10]; (* FIRST 10 CHAR OF FILE NAME *)
LINECOUNT : INTEGER; (* CURRENT NUMBER OF LINES *)
NFILE : TEXT; (* NEXT RECORD FILE *)
POFLAG : INTEGER; (* IF YES- POSTFIX OK *)
QUERY : PQUERY[1..90] OF TOKENS; (* POSTFIX FORM OF QUERY *)
PRINTER : INTERACTIVE; (* PRINTING FILE *)
PRTER : INTEGER; (* IF YES- PRINTER ONLINE *)
RYNT : INTEGER; (* NUMBER CHAR IN QUERY *)
QUERY : PQUERY[1..512] OF CHAR; (* SOURCE FORM OF QUERY *)
RDNCTNT : INTEGER; (* READLNS TO EOF *)
RECNAMPL : STRING[30]; (* PLURAL OF RECORD NAME *)
RECNAMING : STRING[30]; (* SINGULAR OF RECORD NAME *)
RFILE : TEXT; (* FILE TO HOLD RECORD *)
SRCHSTR : PQUERY[STR1..STR15] OF STRING[255]; (* ARRAY OF SEARCH STRINGS *)
TFILE : TEXT; (* FILE TO HOLD RECORD TEMPORARILY *)

PROCEDURE DUMMY;

IMPLEMENTATION

PROCEDURE DUMMY;
BEGIN
END;

BEGIN
BACKSP := CHR(8); (* BACKSPACE *)
BELL := CHR(7); (* BELL *)
CTRLC := CHR(3); (* EOF CHAR *)
DELLIMITER := CHR(04); (* CTRL-D *)
ESC := CHR(27); (* ESCAPE CHARACTER *)
RTARROW := CHR(21); (* RIGHT ARROW (->) *)
(* CHECK FOR PRINTER ON-LINE *)
UNITCLEAR(6);
IF IORESULT = 0
THEN BEGIN
  RESET(PRINTER,'PRINTER:');
  PRTER := YES;
END
ELSE PRTER := NO;
(* FIELD IDENTIFIERS *)
UNIT EDIT;
(* PROVIDES BASIC SCREEN EDITING FUNCTIONS *)
(* OF UP, DOWN, RIGHT, LEFT, CR, ADD, AND *)
(* ERASE. *)

INTERFACE
USES GLOBAL;

PROCEDURE EDITOR(VAR BUFFER: PACKEDARRAY; VAR NUMCHAR:INTEGER;
MESSAGE: STRING);

IMPLEMENTATION
VAR

ADDMGS : STRING[40]; (* LEGAL ADD MODE COMMANDS *)
ANSWER  : CHAR; (* USER RESPONSE *)
CHARCOUNT : INTEGER; (* NUMBER OF CHARS ON CURRENT LINE *)
COLUMN   : INTEGER; (* COLUMN CURSOR IS IN *)
DELMGS  : STRING[40]; (* LEGAL DELETE MODE COMMANDS *)
LEFTMARKER : INTEGER; (* LAST COL WHICH CAN BACK OVER *)
LINE     : INTEGER; (* LINE CURSOR IS ON *)
MOVESGS : STRING[40]; (* LEGAL MOVE MODE COMMANDS *)
NUMBLINES : INTEGER; (* HOW MANY LINES PRINTED ON SCREEN *)
NUMBCHAR : INTEGER; (* NUM CHAR IN FIELD *)
RIGHTMARKER : INTEGER; (* 1ST COL WHICH CAN'T TYPE OVER *)
SCRNPAGE : PACKED ARRAY[1..203] OF PACKED ARRAY[1..80] OF CHAR;
(* INTERNAL - CONTENTS OF SCREEN *)

PROCEDURE CALCCOUNT;
(* CALCULATE NUMBER OF CHARACTERS ON THE CURRENT *)
(* LINE. *)
VAR
DONE   : INTEGER; (* IF YES- CHARCOUNT FOUND. *)
BEGIN
CHARCOUNT := 80; (* START AT RIGHT *)
DONE := NO;
WHILE (CHARCOUNT > 0) AND (DONE = NO) DO
  IF SCRNPAGE[LIN,CHARCOUNT] <> SPACE
    THEN DONE := YES
    ELSE CHARCOUNT := CHARCOUNT - 1;
END; (* CALCCOUNT *)

PROCEDURE RIGHTARROW;
(* THIS PROCEDURE REINSERTS CHARACTERS JUST ERASED *)
(* BY LEFTARROW IN THE ERASE ROUTINE. *)
BEGIN
  IF COLUMN <> 81
THEN
   IF (COLUMN < RIGHTMARKER) OR (RIGHTMARKER = 80)
      THEN BEGIN (* RESTORE CHARACTER *)
         NUMBCHAR := NUMBCHAR + 1;
         CHARCOUNT := CHARCOUNT + 1;
         COLUMN := COLUMN + 1;
         WRITE(OUTPUT, SCRNFPAGE[LINE, COLUMN-1]);
      END
      ELSE WRITE(OUTPUT,BELL)
      ELSE WRITE(OUTPUT,BELL);
   END; (* RIGHTARROW *)

PROCEDURE LEFTARROW;
   (* THIS PROCEDURE DELETES CHARACTERS UP TO THE
      (* LEFT MARKER. *)
   BEGIN
      IF (COLUMN > LEFTMARKER) AND (COLUMN <> 1)
         THEN BEGIN
            IF COLUMN <> 81
               THEN BEGIN
                  WRITE (OUTPUT, BACKSP) ;
                  WRITE (OUTPUT, SPACE) ;
                  WRITE (OUTPUT, BACKSP) ;
               END
               ELSE WRITE (OUTPUT, SPACE);
            NUMBCHAR := NUMBCHAR - 1;
            CHARCOUNT := CHARCOUNT - 1;
            COLUMN := COLUMN - 1;
         END
         ELSE WRITE(OUTPUT,BELL);
      END;
   END: (* LEFTARROW *)

PROCEDURE NEWLINE;
   (* CHECKS TO SEE IF A CARRIAGE RETURN IS ALLOWED *)
   (* AND IF IT IS, GENERATES A CARRIAGE RETURN. *)
   VAR
      COUNT : INTEGER; (* COUNTER *)
   BEGIN
      IF NOT((CHARCOUNT = 80) AND (COLUMN <> 81))
         THEN BEGIN
            IF (RIGHTMARKER = 80) AND (NUMBLINES < 20)
               THEN BEGIN (* OK TO CR *)
                  COLUMN := 1;
                  LINE := LINE + 1;
                  CHARCOUNT := 0;
                  NUMBLINES := NUMBLINES + 1;
                  LEFTMARKER := 1;
               (* MOVE LINES DOWN *)
               FOR COUNT := NUMBLINES DOWNTO LINE+1 DO
                  SCRNFPAGE[LINE] := SCRNFPAGE[COUNT-1];
               (* BLANK CURRENT LINE *)
         END
         ELSE WRITE(OUTPUT, BACKSP) ;
         END;
FILLCHAR(SCRNPAGEELINE3,80,SPACE);
(* IF ANSWER <> BLANK, THEN PUT ANSWER IN PAGE *)
IF ANSWER <> SPACE
THEN BEGIN
  CHARCOUNT := CHARCOUNT + 1;
  SCRNPAGE[LINEN,COLUMN] := ANSWER;
  COLUMN := COLUMN + 1;
  NUMBCHAR := NUMBCHAR + 1;
END;
(* PRINT PAGE *)
GOTOXY(0,0);
FOR COUNT := 1 TO NUMBLINES DO
  WRITELN(OUTPUT,SCRNPAGE[COUNT]);
GOTOXY(COLUMN-1,LINE-1);
ELSE WRITE(OUTPUT,BELL);
END
END;

PROCEDURE SCRUNCH;
(* SCRUNCHES LINE AFTER AN INSERT. *)
VAR
  COUNT : INTEGER; (* COUNTER *)
  CNT : INTEGER; (* COUNTER *)
BEGIN
  IF CHARCOUNT < 80
  THEN BEGIN (* BLANKS TO BE REMOVED FROM LINE *)
    CNT := COLUMN;
    FOR COUNT := RIGHTMARKER TO 80 DO
      BEGIN
        SCRNPAGE[LINEN,CNT] := SCRNPAGE[LINEN,COUNT];
        CNT := CNT +1;
      END;
    FOR COUNT := CNT TO 80 DO
      SCRNPAGE[LINEN,COUNT] := SPACE;
      (* DON'T PAINT SCREEN ON USER GENERATED CR *)
    IF NOT(EOLN(KEYBOARD))
    THEN BEGIN
      GOTOXY(0,0);
      FOR COUNT := 1 TO NUMBLINES DO
        WRITELN(OUTPUT,SCRNPAGE[COUNT]);
      GOTOXY(COLUMN-1,LINE-1);
    END;
  END;
END; (* SCRUNCH *)

PROCEDURE INSERT;
(* INSERTS CHARACTERS INTO EXISTING TEXT *)
BEGIN
  REPEAT (* GET LEGAL CHAR *)
READ(KEYBOARD, ANSWER);
IF ANSWER = RTARROW
THEN WRITE(OUTPUT, BELL);
UNTIL ANSWER <> RTARROW;
WHILE (NOT(EOF(KEYBOARD))) AND (ANSWER <> CTRL) DO
BEGIN
IF ANSWER <> BACKSF
THEN BEGIN
IF NOT(EOLN(KEYBOARD))
THEN BEGIN (* NOT CR *)
IF (COLUMN <= RIGHTMARKER) AND (CHARCOUNT < 80)
THEN BEGIN
IF NUMBCHAR < 492
THEN BEGIN (* ADD CHAR *)
WRITE(OUTPUT, ANSWER);
CHARCOUNT := CHARCOUNT + 1;
IF CHARCOUNT = 75
THEN WRITE(OUTPUT, BELL);
SCRNPAGEELINE, COLUMN := ANSWER;
COLUMN := COLUMN + 1;
NUMBCHAR := NUMBCHAR + 1;
END
ELSE WRITE(OUTPUT, BELL);
END ELSE NEWLINE;
END ELSE BEGIN
SCRUNCH;
NEWLINE;
END;
END
ELSE LEFTARROW;
REPEAT (* GET LEGAL CHAR *)
READ(KEYBOARD, ANSWER);
IF ANSWER = RTARROW
THEN WRITE(OUTPUT, BELL);
UNTIL ANSWER <> RTARROW;
END;
RESET(KEYBOARD);
END; (* INSERT *)

PROCEDURE ADDCHAR;
(* PREPARES FOR INSERTING CHARACTERS INTO TEXT. *)
VAR
  COUNT : INTEGER; (* COUNTER *)
BEGIN
  GOTOXY(0, 21);
  WRITELN(OUTPUT, '"<<<<ADD CHARACTER>>>"');
  WRITE(OUTPUT, ADDMSG);
  GOTOXY(COLUMN-1, LINE-1);
  IF (CHARCOUNT <> 80) OR (COLUMN > 80)
  THEN BEGIN (* INSERT ALLOWED *)
(* PREPARE FOR INSERT *)
LEFTMARKER := COLUMN;
RIGHTMARKER := 80;
(* MOVE LINE RIGHT OF COLUMN TO COL 80 *)
IF (CHARCOUNT > 0) AND (CHARCOUNT < 80) AND (COLUMN = CHARCOUNT)
THEN BEGIN
FOR COUNT := CHARCOUNT DOWNTO COLUMN DO
BEGIN
SCRNPAGE(LINE,RIGHTMARKER) := SCRNPAGE(LINE,COUNT);
RIGHTMARKER := RIGHTMARKER - 1;
END;
RIGHTMARKER := RIGHTMARKER + 1;
FOR COUNT := RIGHTMARKER - 1 DOWNTO LEFTMARKER DO
SCRNPAGE(LINE,COUNT) := SPACE;
GOTOXY(0,LINE-1);
WRITE(OUTPUT,SCRNPAGE(LINE));
GOTOXY(COLUMN-1,LINE-1);
END;
(* INSERT CHAR *)
INSERT;
(* SCRUNCH UP LINE *)
SCRUNCH;
END ELSE WRITE(OUTPUT,BELL);
GOTOXY(0,21);
WRITELN(OUTPUT,'<<<<ERASE CHARACTER>>>>');
WRITE(OUTPUT,DELMSG);
GOTOXY(COLUMN-1,LINE-1);
END; (* ADDCHAR *)

PROCEDURE ERASECHAR;
(* DELETES CHARACTERS FROM A LINE UNTIL THE LEFT *)
(* EDGE OF THE LINE IS REACHED. *)
VAR
CNT : INTEGER; (* LOOP COUNTER. *)
COUNT : INTEGER; (* LOOP COUNTER. *)
BEGIN
GOTOXY(0,21);
WRITELN(OUTPUT,'<<<<ERASE CHARACTER>>>>');
WRITE(OUTPUT,DELMSG);
GOTOXY(COLUMN-1,LINE-1);
LEFTMARKER := 1;
RIGHTMARKER := COLUMN;
REPEAT (*) GET INPUT (*)
READ(KEYBOARD,ANSWER);
IF NOT(ANSWER IN [BACKSP,RTARROW]) THEN IF (NOT (EOF(KEYBOARD))) AND (ANSWER <> CTRLC)
THEN WRITE(OUTPUT,BELL);
UNTIL (ANSWER IN [BACKSP,RTARROW]) OR (EOF(KEYBOARD))
OR (ANSWER = CTRLC);
WHILE (NOT(EOF(KEYBOARD))) AND (ANSWER <> CTRLC) DO
BEGIN
IF ANSWER = BACKSP
    THEN LEFTARROW (* BACKSPACE - ERASE *)
ELSE RIGHTARROW; (* RIGHTARROW - RECALL *)
REPEAT (* GET INPUT *)
READ (KEYBOARD, ANSWER);
    IF NOT (ANSWER IN [BACKSP,RTARROW])
        THEN IF (NOT (EOF (KEYBOARD))) AND (ANSWER <> CTRLC)
            THEN WRITE (OUTPUT, BELL);
        UNTIL (ANSWER IN [BACKSP,RTARROW]) OR (EOF (KEYBOARD))
            OR (ANSWER = CTRLC);
END;
(* CLOSE UP GAP LEFT BY DELETES IF THERE IS ONE. *)
IF (COLUMN < RIGHTMARKER)
    THEN BEGIN
        CNT := COLUMN;
        FOR COUNT := RIGHTMARKER TO 80 DO
            BEGIN
                SCRNPAGE[LINEX, CNT] := SCRNPAGE[LINEX, COUNT];
                CNT := CNT + 1;
            END;
        FOR COUNT := CNT TO 80 DO
            SCRNPAGE[LINEX, COUNT] := SPACE;
        GOTOXY(OLINEX - 1);
        WRITE (OUTPUT, SCRNPAGE[LINEX]);
        GOTOXY(0, 21);
        WRITELN (OUTPUT, '<<<< MOVE CURSOR >>>>>');
        WRITE (OUTPUT, MOVEMSG);
        GOTOXY(COLUMN - 1, LINE - 1);
        RESET (KEYBOARD);
    END; (* ERASECHAR *)
PROCEDURE UP;
    (* CAUSES THE CURSOR TO MOVE UP ONE LINE UNTIL *)
    (* THE TOP OF THE PAGE IS ENCOUNTERED. *)
BEGIN
    IF LINE <> 1
        THEN BEGIN (* CAN MOVE UP *)
            LINE := LINE - 1;
            CALLCOUNT;
            IF (CHARCOUNT < COLUMN) AND (COLUMN <> 1)
                THEN COLUMN := CHARCOUNT;
            GOTOXY(COLUMN - 1, LINE - 1);
        END
    ELSE WRITE (OUTPUT, BELL);
END; (* UP *)
PROCEDURE RETURN;
    (* ALLOWS THE CURSOR TO MOVE DOWN ONE LINE UNTIL *)
    (* THE LAST LINE OF TEXT IS REACHED. *)
BEGIN
IF LINE <> 20
    THEN BEGIN (* NOT AT BOTTOM OF PAGE *)
        IF LINE < NUMBLINES
            THEN BEGIN (* CAN GO DOWN ONE *)
                LINE := LINE + 1;
                CALCOUNT;
                GOTOXY(COLUMN-1,LINE-2);
                COLUMN := 1;
            END ELSE WRITE(OUTPUT,BELL);
        END ELSE WRITE(OUTPUT,BELL);
    END; (* RETURN *)

PROCEDURE DOWN;
    (* ALLOWS THE CURSOR TO MOVE DOWN ONE LINE UNTIL *)
    (* THE LAST LINE OF TEXT IS REACHED. *)
BEGIN
    IF LINE <> 20
        THEN BEGIN (* NOT AT BOTTOM OF PAGE *)
            IF LINE < NUMBLINES
                THEN BEGIN (* CAN GO DOWN ONE *)
                    LINE := LINE + 1;
                    CALCOUNT;
                    IF (CHARCOUNT < COLUMN) AND (COLUMN <> 1)
                        THEN COLUMN := CHARCOUNT;
                    GOTOXY(COLUMN-1,LINE-1);
                END ELSE WRITE(OUTPUT,BELL);
            END ELSE WRITE(OUTPUT,BELL);
        END; (* DOWN *)

PROCEDURE LEFT;
    (* CAUSES THE CURSOR TO MOVE ONE SPACE LEFT *)
    (* UNTIL THE BEGINNING OF LINE IS REACHED. *)
BEGIN
    IF COLUMN <> 1
        THEN BEGIN (* CAN MOVE LEFT *)
            IF COLUMN <> 8
                THEN WRITE(OUTPUT,BACKSP);
            COLUMN := COLUMN - 1;
        END ELSE BEGIN
            IF LINE <> 1
                THEN BEGIN
                    LINE := LINE - 1;
                    CALCOUNT;
                    COLUMN := CHARCOUNT;
                    GOTOXY(COLUMN-1,LINE-1);
                END
ELSE WRITE (OUTPUT, BELL);
END;
END; (* LEFT *)

PROCEDURE RIGHT;
(* CAUSES THE CURSOR TO MOVE ONE SPACE RIGHT *)
(* UNTIL THE END OF THE CHARACTERS IS REACHED. *)
BEGIN
  IF COLUMN <= CHARCOUNT THEN BEGIN (* CAN MOVE RIGHT *)
    GOTOXY (COLUMN, LINE-1);
    COLUMN := COLUMN + 1;
  END ELSE WRITE (OUTPUT, BELL);
END; (* RIGHT *)

PROCEDURE EDITOR; (* (VAR BUFFER: PACKEDARRAY; VAR NUMCHAR: INTEGER; MESSAGE: STRING(80)) *)
(* THIS EDITOR ALLOWS INPUT ON A SINGLE 20 LINE *)
(* BY 80 CHAR SCREEN UNTIL 512 CHARS HAVE BEEN *)
(* ENTERED, OR A CTRL-C IS ENTERED. *)
VAR
  COUNT : INTEGER; (* LOOP COUNTER. *)
BEGIN
  (* SETUP *)
  FILLCHAR (SCANPAGE[1J, 1600, SPACE);
  NUMCHAR := NUMCHAR;
  PAGE (OUTPUT);
  NUMBLINES := 1;
  IF NUMCHAR <> 0 THEN BEGIN (* THERE ARE CHAR IN BUFFER *)
    LINE := 1; (* ON FIRST LINE. *)
    CHARCOUNT := 1; (* READY TO TRANSFER 1ST CHAR *)
    FOR COUNT := 1 TO NUMCHAR DO
      BEGIN
        IF BUFFER[COUNT] = DELIMITER THEN BEGIN (* NEW LINE *)
          NUMBLINES := NUMBLINES + 1;
          LINE := LINE + 1;
          CHARCOUNT := 1;
        END ELSE BEGIN (* SAME LINE *)
          SCRNPAGE[LINEL CHARCOUNT] := BUFFER[COUNT];
          CHARCOUNT := CHARCOUNT + 1;
        END;
      END;
  END;
  (* PRINT PAGE *)
  FOR LINE := 1 TO NUMBLINES DO
    WRITELN (OUTPUT, SCRNPAGE[LINEL]);
END;
(* PRINT BOTTOM MESSAGE *)
GOTOXY(0,20);
WRITE(OUTPUT,'-------------1-----------2-----------3-----------4');
WRITELN(OUTPUT,'-------------5-----------6-----------7-----------8');
WRITELN(OUTPUT,'<<<<: MOVE CURSOR >>>>>');
WRITE(OUTPUT,MOVEMSG);
MESSAGE := CONCAT('--->',MESSAGE);
WRITE(OUTPUT,MESSAGE);
GOTOXY(0,0);
COLUMN := 1; (* CURSOR IN COLUMN 1 *)
LINE := 1; (* CURSOR ON LINE 1 *)
(* CALCULATE CHARCOUNT OF LINE ONE *)
CALCOUNT;
(* GET USER INPUT *)
REPEAT (* UNTIL CTRL-C IS INPUT *)
READ KEYBOARDR, ANSWER);
IF NOT(ANSWER IN 'A', 'a', 'E', 'e', 'I', 'i', 'M', 'm', 'J', 'j', 'K', 'k', 'BACKSP', 'RTARROW', 'SPACE') THEN BEGIN
  IF NOT(ANSWER IN 'Q', 'q')
  THEN WRITE(OUTPUT,BELL);
END ELSE
CASE ORD(ANSWER) OF
  65,97 (* 'A', 'a' *) : ADDCHAR;
  69,101 (* 'E', 'e' *) : ERASECHAR;
  73,105 (* 'I', 'i' *) : UP;
  77,109 (* 'M', 'm' *) : DOWN;
  8,74,106 (* BACKSP,'J', 'j' *) : LEFT;
  21,75,107 (* RTARROW,'K', 'k' *) : RIGHT;
  32 (* SPACE *) : RETURN; (*CR TRANS TO SPACE *)
END;
UNTIL ANSWER IN ['Q', 'q'];
(* END EDITOR *)
(* WRITE PAGE TO BUFFER *)
PAGE(OUTPUT);
GOTOXY(0,20);
WRITELN(OUTPUT,'PLEASE WAIT - ':14,' ':80);
NUMBCHAR := 0;
FOR LINE := 1 TO NUMBLINES DO
BEGIN
  (* CALCULATE CHARCOUNT OF LINE *)
  CALCOUNT:
  IF CHARCOUNT <> 0
  THEN BEGIN (* PAGE HAS CHARACTERS *)
    FOR COUNT := 1 TO CHARCOUNT DO
      BEGIN
        NUMBCHAR := NUMBCHAR + 1;
        BUFFER[NUMBCHAR] := SCANPAGE(LINE,COUNT);
      END;
    NUMBCHAR := NUMBCHAR + 1;
    BUFFER[NUMBCHAR] := DELIMITER;
  END;
END;
NUMCHAR := NUMBCHAR;
PAGE(OUTPUT); LINECOUNT := 0;
END; (* EDITOR *)
BEGIN
(* INITIALIZE *)
MOVEMSG := 'A)DD E)RASE Q)UIT I,M,J,K,<->,RETURN';
ADDMSG := '<-BACKSPACE CNTL-C: END ADD ';
DELMMSG := '<-ERASE ->:RESTORE CTRL-C:END ERASE ';
BIBLIOGRAPHY

Books


Articles

Public Documents


Database System -- DB Master, Stoneware Microcomputer Products, San Rafael, California, June, 1981.


Unpublished Materials

Creech, Teresa A., Information Storage and Retrieval and Microcomputers, North Texas State University, November 18, 1980.